UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Reducción de dimensionalidad aplicada a imágenes de satélite

José Manuel Chacón Aguilera

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Reducción de dimensionalidad aplicada a imágenes de satélite

Autor: José Manuel Chacón Aguilera
Tutor: José Ramón Dorronsoro Ibero

junio 2021

José Manuel Chacón Aguilera
Reducción de dimensionalidad aplicada a imágenes de satélite

José Manuel Chacón Aguilera
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi madre, padre y hermanos.

La ciencia se compone de errores,
que a su vez son los pasos hacia la verdad.

Julio Verne

Agradecimientos

En primer lugar me gustaría agradecer a mi familia, por confiar en mí y apoyarme incondicionalmente durante los cinco años de carrera. Vuestro cariño y apoyo me han ayudado mucho en los momentos más difíciles.

En segundo lugar, muchas gracias a mis compañeros; Alejandro, Cabornero, Sergio, Miguel, Mario y Rafa. Las tardes de estudio con vosotros y la puesta en común de nuestras dudas e inquietudes han sido una de las claves para llegar hasta aquí.

Finalmente quiero dedicar unas palabras a mi tutor. Jose, gracias a tu paciencia y sabiduría no solo ha sido posible este proyecto sino que he aprendido una gran cantidad de cosas nuevas. Muchas gracias también por acompañarme durante los cinco años de carrera y orientarme cuando estaba más perdido.

Resumen

En este proyecto se estudiarán diversos métodos de reducción de dimensionalidad con la finalidad de aplicarlos a imágenes de satélite. Es interesante aplicar estas técnicas a los datos mencionados para conseguir un ahorro de recursos tanto en el almacenamiento como en el procesamiento de los mismos. Para conseguir este objetivo debemos primero obtener los datos objeto de este estudio; para lo cual exploraremos dos agencias europeas que proporcionan información meteorológica de toda la superficie terrestre (aunque en este proyecto nos centraremos en las condiciones meteorológicas de la península ibérica). En este estudio de las fuentes comprenderemos de qué modo obtienen las agencias estos datos y la utilidad de los mismos. En segundo lugar estudiaremos diversos algoritmos de reducción de dimensionalidad, viendo la base teórica y su funcionamiento. En esta parte del proyecto estudiaremos los algoritmos PCA, ICA y NMF; además de dos tipos de autoencoders (lineal y multicapa), una arquitectura de red neuronal que comprime los datos que recibe forzándolos a pasar por un cuello de botella. Como preámbulo para el objetivo final de este proyecto realizaremos experimentos sobre dos bases de datos consistentes en imágenes para comprobar la eficacia de los algoritmos de reducción de dimensionalidad estudiados. Finalmente aplicaremos estas técnicas de reducción a los datos meteorológicos obtenidos y también modelaremos la predicción de la evolución temporal de estos datos tanto en las imágenes originales como en su representación reducida.

Palabras clave

Reducción de dimensionalidad, PCA, datos meteorológicos, cuello de botella, autoencoder, red neuronal, predicción de la evolución temporal.

Abstract

In this project, various dimensionality reduction methods will be studied in order to apply them to satellite images. It is interesting to apply these techniques to meteorological data in order to save resources both in storage and in their processing. To achieve this objective we must first obtain the data object of this study; for which we will explore two European agencies that provide meteorological information of the entire earth's surface (although in this project we will focus on the meteorological conditions of the Iberian Peninsula). In the study of the sources we will understand how agencies obtain these data and the utility of them. Second, we will study various dimensionality reduction algorithms, looking at their theoretical basis and how they work. In this part of the project we will study the PCA, ICA and NMF algorithms; in addition we will also study two types of autoencoders (linear and multilayer), a neural network architecture that compresses the data it receives, forcing it to go through a bottleneck. As a preamble to the final objective of this project, we will carry out experiments on two databases consisting of images; to verify the effectiveness of the dimensionality reduction algorithms studied. Finally, we will apply these reduction techniques to the meteorological data obtained and we will also model the prediction of the temporal evolution of these data both in the original images and in their reduced representation.

Keywords

Dimensionality reduction, PCA, meteorological data, bottleneck, autoencoder, neural network, prediction of temporal evolution.

Índice

Listas

Lista de figuras

Lista de tablas

1

Introducción

La reducción de dimensionalidad es la transformación de datos de un espacio de alta dimensión a un espacio de baja dimensión de modo que la representación de baja dimensión conserve una gran parte de la información original. Trabajar con datos con una gran cantidad de atributos puede resultar indeseable por muchas razones; el análisis de los datos brutos suele ser difícil de resolver desde el punto de vista computacional como consecuencia de la maldición de la dimensionalidad; es decir, el número de atributos hace crecer de forma exponencial la cantidad de datos necesarios para el entrenamiento de modelos y por tanto crece también la cantidad de recursos computacionales necesarios. La reducción de la dimensionalidad es común en campos que se ocupan de procesar datos con un gran número de observaciones y / o un gran número de variables, como el procesamiento de señales, el reconocimiento de voz, la neuroinformática etc. . . . En este proyecto analizaremos el campo de la meteorología; concretamente la información obtenida por imágenes de satélite.

1.1. Objetivo general

En primer lugar, el objetivo es conocer las agencias que estudian la meteorología, obtener datos de las mismas y procesar un dataframe para operar con ellos. Dentro de este proceso queremos comprender qué información aportan los diferentes productos disponibles, de qué modo viene dada dicha información y cuál es su utilidad. En segundo lugar, estudiaremos la teoría de diversos métodos de reducción de dimensionalidad para saber cómo es el funcionamiento de los mismos. Estamos interesados en estudiar algoritmos que tengan un método de compresión y otro de descompresión; para obtener la reconstrucción de los datos originales a partir de su representación reducida.

El objetivo general de este proyecto será aplicar diversas técnicas de reducción de dimensionalidad a datos consistentes en imágenes simuladas de satélite. Estas imágenes proporcionan información sobre el clima y las condiciones meteorológicas de la superficie terrestre; en concreto en este estudio se observará una variable que mide la cantidad de radiación reflejada por la superficie. Sin embargo, para tener suficiente información sobre una zona concreta debemos recopilar una gran cantidad de imágenes correspondientes a múltiples instantes de tiempo. Almacenar y procesar datos correspondientes a

largos periodos de tiempo es costoso y esto motiva el aplicar técnicas de reducción de dimensionalidad a estos datos. En este proyecto nos vamos a restringir al estudio de las condiciones meteorológicas de la península ibérica durante los años 2019 y 2020.

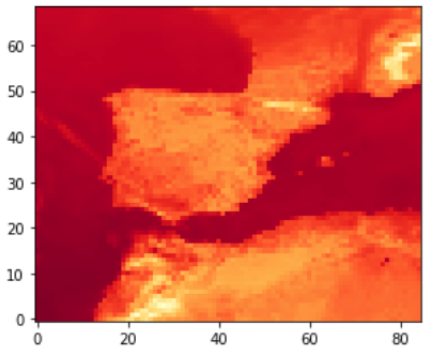

Figura 1.1: Área estudiada, primera hora del año 2019.

Estas imágenes tienen una resolución de 85×69 píxeles; lo cual hace un total de 5865 variables. En este proyecto estudiaremos métodos que reduzcan la dimensión a 300 variables, lo cual supone una compresión del 95 % aproximadamente.

El objetivo final de realizar esta reducción a los datos es aplicar técnicas de predicción de la evolución temporal sobre las imágenes originales y también sobre su representación reducida y comprobar cuál es la calidad de la predicción en ambos casos. Operar con datos cuyas variables han sido reducidas en un 95 % supone un gran ahorro computacional, tanto en términos de memoria como de procesamiento. Además, el estudio y la predicción de la *radiación reflejada* tiene aplicaciones directas en el campo de las energías renovables ya que conocer las zonas que reflejan mayor cantidad de radiación es útil para la instalación de paneles solares que aprovechen esa energía.

1.2. Estructura de la memoria

Vamos a estructurar esta memoria en 4 capítulos (a partir de este capítulo introductorio); primero nos centraremos en las fuentes donde hemos obtenido los datos, posteriormente estudiaremos diversos algoritmos de reducción de dimensionalidad, explicaremos los experimentos realizados y para finalizar dedicaremos un capítulo a las conclusiones y el trabajo futuro.

En primer lugar, estudiaremos las fuentes que proporcionan datos meteorológicos y analizaremos los diferentes productos ofrecidos por dos grandes agencias. Dividiremos este capítulo en dos secciones principales; en la primera estudiaremos el *European Organisation for the Exploitation of Meteorological Satellites* (EUMETSAT) [1] y en la segunda sección estudiaremos el *European Centre for Medium-Range Weather Forecasts* (ECMWF) [2]. De esta última agencia haremos énfasis en las imágenes simuladas de satélite, que son los datos que usaremos posteriormente para nuestros experimentos.

En segundo lugar vamos a dedicar un capítulo a estudiar la teoría de varios algoritmos de reducción de dimensionalidad. En este capítulo se estudiarán los algoritmos PCA, NMF e ICA; además estudiaremos cómo funciona una red neuronal y de qué forma podemos usar esta técnica computacional para programar dos arquitecturas de autoencoder; uno lineal y otro multicapa. Un autoencoder es una red que reconstruye la entrada recibida haciendo pasar los datos por un cuello de botella (una capa con un número de neuronas menor a la entrada por donde se fuerzan a pasar los datos originales para comprimirlos).

El cuarto capítulo lo dedicaremos a realizar experimentos. Hemos estructurado este capítulo en dos secciones; la primera sección la dedicamos a probar los algoritmos de reducción de dimensionalidad estudiados sobre las bases de datos *Olivetti's Faces* [3] y *MNIST* [4]. Estos datos consisten en imágenes representando caras de personas y números dibujados sobre fondo blanco respectivamente; por tanto esperamos que los resultados obtenidos sean reproducibles en las imágenes de satélite. La segunda sección consiste en comprobar la eficacia de los algoritmos de reducción de dimensionalidad aplicados las imágenes de satélite y en realizar predicciones de la evolución temporal de las mismas.

2

Estudio de las fuentes de datos

Para entrenar nuestro modelo necesitamos datos meteorológicos de la península ibérica. Para la obtención de los mismos hemos tenido que investigar qué agencias ofrecen los datos de forma gratuita con fines de investigación. Principalmente hemos estudiado dos fuentes, el *European Organisation for the Exploitation of Meteorological Satellites* (EUMETSAT) [1] y el *European Centre for Medium-Range Weather Forecasts* (ECMWF) [2] que son dos agencias europeas dedicadas a la meteorología.

Estamos interesados en obtener información de la radiación sobre la superficie terrestre y localizar las zonas con mayor temperatura y menor concentración de nubes. Estudiaremos las variables ofrecidas por el producto *Spinning Enhanced Visible and InfraRed Imager* (SEVIRI) [5] de Eumetsat aunque la estructura compleja de este producto nos ha imposibilitado (como comentaremos más adelante) procesar un dataframe. Para el proyecto usaremos la predicción de la variable *Cloudy Brightness Temperature* durante los años 2019 y 2020, ofrecida por el ECMWF. Esta variable nos da información sobre la temperatura de las nubes, lo cual además de información sobre la posible meteorología dependiendo de la temperatura nos dice qué zonas tienen una concentración de nubes menor.

2.1. EUMETSAT

El EUMETSAT es una agencia europea que opera con datos satelitales para la monitorización del tiempo atmosférico, el clima (es decir el tiempo atmosférico a lo largo de periodos considerables de tiempo) y también las condiciones que puedan ocurrir en el espacio [6]. Está compuesta por 30 estados miembros, todos ellos localizados en Europa. Aún así mantiene estrechas relaciones con otras agencias espaciales como la NASA.

Como una agencia global dedicada a la meteorología, entre sus labores principales se encuentran las siguientes: el desarrollo y construcción de nuevos satélites; el procesamiento de datos de los satélites que actualmente están orbitando; el estudio de la composición de la atmósfera y la distribución de estos datos a sus usuarios.

Esta agencia ofrece una resolución temporal de los datos de hasta 15 minutos. Además de tener una muy buena resolución espacial (hasta 0.03 grados) y una gran cantidad de productos disponibles.

2.1.1. Detalles de acceso y recepción de los datos

Para el acceso a los datos la agencia ha desarrollado un portal web donde los usuarios pueden darse de alta para explorar los diferentes productos ofrecidos. Para poder acceder al EUMETSAT DATA CENTRE [7] necesitamos crear una cuenta gratuita. Con esta cuenta tenemos acceso a una gran cantidad de productos, aunque con ciertas limitaciones en los tamaños de descarga y la cantidad de peticiones simultáneas.

Una vez dados de alta en el DATA CENTRE podemos enviar una petición de datos. El proceso de enviar una petición está guiado por una aplicación web. En dicha aplicación iremos «filtrando» cuales son los datos de nuestro interés.

Comenzamos seleccionando los productos que nos interesan y los satélites de los cuales queremos obtener dicha información. Posteriormente se nos abre una ventana para seleccionar la franja temporal en la que estamos interesados así como la resolución temporal. Tras este paso ya nos indica el tamaño aproximado de los datos que vamos a descargar. Los pasos restantes consisten en la selección de la zona a estudiar; este paso es bastante intuitivo ya que consiste en dibujar un recuadro sobre la zona del mapamundi correspondiente, posteriormente se nos abre un menú para seleccionar el formato y para finalizar elegimos el modo de recepción de los datos y si queremos recibirlos comprimidos o no (también podemos elegir el modo de compresión). Para la recepción de los datos podemos elegir entre el formato físico o «recepción en linea»; hemos optado por la segunda opción por ser más cómoda (además de gratuita). Una vez dichos datos están en linea nos envían un correo con las instrucciones para su descarga. En función del volumen de datos pedido, el correo con las instrucciones de descarga tardará más o menos, estando siempre entre un par de días y un par de semanas.

No fuimos capaces de procesar correctamente los datos obtenidos debido a dos causas. El menú de selección de franja horaria y resolución temporal era bastante confuso, además para seleccionar solamente ciertos canales (de los 12 que ofrece SEVIRI) necesariamente teníamos que seleccionar el formato NetCDF y no conseguimos procesar un dataframe ya que no conseguimos «descifrar» la organización que seguían los ficheros descargados.

2.1.2. Productos disponibles

Su red de satélites está compuesta por los siguientes, aunque el programa MTG con sus 6 satélites geo-estacionarios aún no se ha lanzado y el primer satélite tiene fecha de lanzamiento para finales del 2022.

1.– Meteosoat First Generation (MFG), 1977-2017 ; 7 satélites geo-estacionarios.

2.– Meteosat Second Generation (MSG), 2004-2025; 4 satélites geo-estacionarios.

3.– Meteosat Third Generation (MTG), 2022-2039; 6 satélites geo-estacionarios.

4.– Metop, 2007-2024; 3 satélites polares.

5.– EUMETSAT Polar System Second Generation (EPS-SG); 2 satélites polares.

6.– Jason, 2009-2039; 3 satélites marítimos.

Nos interesamos en el producto *Spinning Enhanced Visible and InfraRed Imager*, (SEVIRI) [5] por sus siglas en inglés. Es uno de los principales instrumentos del MSG. Su uso es la observación de la superficie terrestre a través de 12 canales distintos. 8 de estos canales son térmicos infrarrojos que registran la radiación reflejada de la superficie terrestre, nubes y superficie oceánica. Otro es el HRV (High Resolution Visible), un canal visible con una resolución de 1km, lo cual mejora los 3km de productos anteriores. Conocer los lugares más «calientes» es útil para el estudio de la instalación de paneles solares, ya que los lugares más calientes se suelen corresponder (a pesar de factores externos) con lugares que reciben gran cantidad de luz solar. El resto de canales registran información sobre la absorción de dióxido de carbono, ozono y vapor de agua.

Satélites geoestacionarios

Estamos interesados en satélites geoestacionarios para obtener información de una zona localizada de la tierra (en nuestro caso la península Ibérica). El lanzamiento de MTG está previsto a lo largo de 2022 y por tanto no hay aún información disponible. Por tanto nos interesamos en los satélites geo-estacionarios de MFG y MSG.

Los satélites artificiales geoestacionarios tiene la propiedad de orbitar sobre el ecuador con la misma velocidad angular que la rotación terrestre, es decir, permanecen «inmóviles» sobre un punto concreto de la tierra. Un satélite de estas características puede «cubrir» hasta un $40\,\%$ de la superficie terrestre dependiendo de su resolución.

Para mantener una órbita geoestacionaria es necesario equilibrar la velocidad angular del satélite con la fuerza gravitatoria ejercida por la propia tierra, de modo que la fuerza centrífuga se equilibre con la gravitatoria. Por tanto cuanto más cerca de la tierra se encuentre el satélite mayor velocidad necesita para que orbite. Los parámetros para satélites geoestacionarios sobre el ecuador son unos 36.000 km sobre la superficie terrestre a una velocidad de 11.000 km/h. La velocidad está fijada en función de la altura (ley de la gravitación universal de Newton), ya que como hemos comentado anteriormente debe ser la misma que la rotación de la tierra.

2.2. ECMWF

El *European Centre for Medium-Range Weather Forecasts* (ECMWF) [2] por sus siglas en inglés, es una institución internacional independiente compuesta por 34 estados miembros. Es tanto un centro de investigación como un servicio operacional que está operativo 24/7 y proporciona predicciones numéricas del tiempo atmosférico a sus estados miembros.

Las lineas de acción principales de este centro son las siguientes:

1.– Producción de predicciones numéricas del tiempo y monitorizar el sistema terrestre.

2.– Investigación técnica y científica para mejorar la precisión de las predicciones.

3.– El mantenimiento y archivo de los datos meteorológicos obtenidos históricamente.

Nosotros veremos cómo se predice la variable *cloudy brightness temperature* donde el término «brightness temperature» es original de la física de procesos radiativos. Representa la temperatura «virtual» a la que un objeto (en nuestro caso las nubes de la península ibérica) emitirían la radiación que es capturada por el satélite. Suponemos que el objeto es un cuerpo negro, es decir, un cuerpo ideal que absorbe toda la radiación que incide sobre él, por tanto; ni refleja ni deja pasar la radiación a través del mismo. Las mediciones realizadas con esta técnica no son totalmente fieles a la realidad, ya que las nubes no son exactamente cuerpos negros.

2.2.1. Predicción numérica del tiempo atmosférico

Como ya comentamos en la sección anterior, no pudimos procesar correctamente un dataframe basándonos en los datos proporcionados por el EUMETSAT. Los datos anteriores se correspondían con «instantáneas» del satélite tomadas en cierto tiempo, lo cual son datos totalmente fieles a los originales, mientras que los datos producidos a raíz de la predicción numérica son el resultado de aplicar un modelo matemático a una observación y generar a partir de dicha observación datos correspondientes al futuro cercano.

La predicción numérica del tiempo incluye varias disciplinas. La asimilación de los datos, la creación del modelo, la estimación de los posibles errores cometidos y el acoplamiento de todas las variables para conformar un sistema más complejo [8].

Asimilación de los datos

Para realizar una predicción necesitamos primero conocer cuál es el punto de partida tanto de la atmósfera como de la superficie terrestre. La calidad de las predicciones va a depender fuertemente de cómo se usen los datos recibidos en tiempo real por el sistema observacional global, el cual consiste en numerosos satélites, estaciones atmosféricas y otros instrumentos.

La finalidad de esta disciplina es determinar el mejor o más probable posible estado de la atmósfera usando observaciones y predicciones a corto plazo. Es un procedimiento secuencial en función del tiempo; un modelo previo se compara con las nuevas observaciones recibidas para actualizar el estado del mismo de modo que dichas observaciones queden reflejadas.

Creación del modelo

Todas las predicciones atmosféricas y re-análisis (el proceso descrito anteriormente mediante el cual a un modelo se le añaden nuevos datos) usan un modelo numérico para realizar una predicción. El ECMWF ha desarrollado su propio modelo atmosférico y su propio sistema de asimilación de los datos, su nombre es el *Integrated Forecasting System* (IFS) por sus siglas en inglés.

Cualquier predicción meteorológica está limitada por el hecho de que la atmósfera es un sistema caótico (nunca podemos conocer exactamente el estado inicial de la atmósfera) y además los modelos matemáticos no pueden representar perfectamente las leyes físicas que rigen los cambios atmosféricos. Esto implica que todas las predicciones van a ir asociadas a cierta incertidumbre. También se debe simplificar los modelos de muchos procesos que ocurren a escalas pequeñas, como la formación de las nubes.

Corrección de errores y fiabilidad del modelo

La atmósfera es un sistema caótico; esto significa que cambios muy ligeros en las condiciones iniciales pueden producir variaciones muy notables al cabo de cierto tiempo [9]. Esto se conoce popularmente como el «efecto mariposa». Se debe de tener en cuenta este efecto y conocer cuál va a ser la incertidumbre al cabo del tiempo.

Para cuantificar cómo de caótico es un estado atmosférico concreto la agencia usa un conjunto de N predictores; esto consiste en ejecutar el modelo N veces cambiando ligeramente las condiciones iniciales en cada ejecución. Tras el paso del tiempo podemos comparar los N resultados y cuantificar cómo de estable ha sido nuestro modelo en función de los datos recibidos y los posibles errores cometidos en la observación. Se suelen usar 50 predictores.

2.2.2. Simulaciones de datos de Satélite (SSD)

Los datos simulados de satélite producidos en el ECMWF proporcionan información sobre la salida del modelo de predicción numérica del tiempo atmosférico. Podemos usar las imágenes simuladas generadas para obtener información sobre las características de las nubes y la humedad de la atmósfera en cierta región. Estas imágenes también se usan con fines de investigación, en particular para validar los desarrollos de modelos que afectan a la humedad y la cobertura de nubes [10].

Los datos satelitales simulados (SSD) se generan usando el modelo predictivo de alta resolución desarrollado por el ECMWF (el *IFS cy41r2*), de la salida de los ciclos 00 UTC y 12 UTC y el mismo modelo de transferencia radiativa rápida que se usa en la asimilación de datos operacionales. La salida de los modelos de pronóstico de alta resolución se utilizan como entrada para calcular variables como la *cloudy brightness temperature*, usando los perfiles del modelo atmosférico relevantes (es

decir, temperatura, humedad específica, cobertura de nubes, contenido específico de agua líquida en la nube, etc...); además de parámetros relevantes de la superficie terrestre (es decir, temperatura de la superficie, componentes de viento, temperatura a 2m, etc...). Se pueden visualizar los datos para construir una secuencia de imágenes generadas por pronóstico que muestren la evolución de las características de las nubes o la humedad derivada del modelo en intervalos de 3 o 6 horas como si fuese datos reales de satélite. En nuestro caso estudiaremos la evolución de la variable *cloudy brightness temperature* sobre la península ibérica en intervalos de 3 horas.

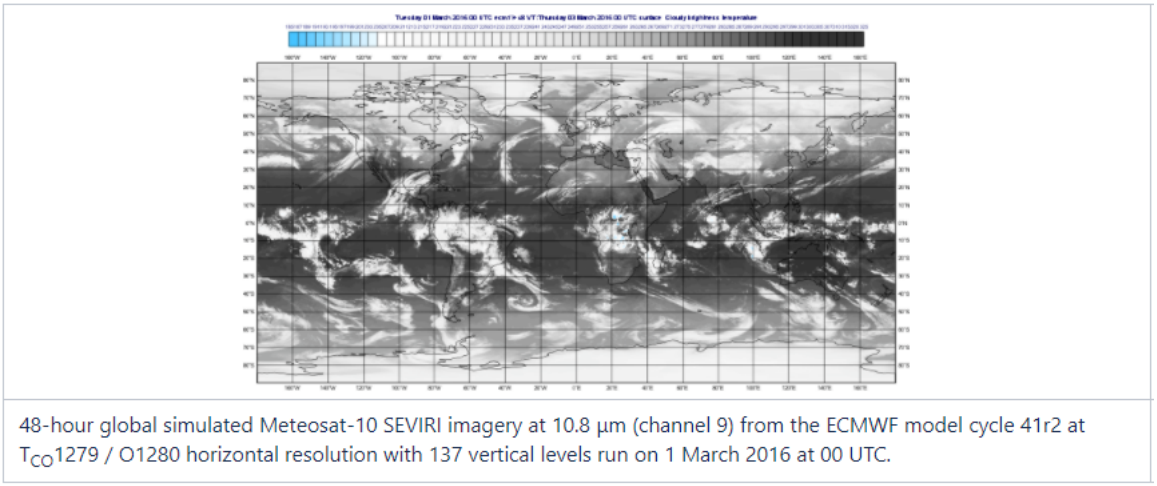

Figura 2.1: Canal 9 de la simulación ejecutada el 1 de marzo de 2016.

Dentro de estos datos existen diversos canales, que dependen de la longitud de onda de las señales recibidas. Nosotros vamos a estudiar el canal 9, con una longitud de onda de $10{,}8\mu m$. Este canal no es visible y se sitúa en el espectro infrarrojo. Las imágenes de satélite simuladas se generan con la finalidad de imitar imágenes de satélite reales. Las simulaciones de satélites geoestacionarios en canales de vapor de agua han sido codificadas utilizando GRIB edición 1, archivadas en MARS y presentadas en el sitio web de ECMWF como archivos de imagen.

La figura 2.1 muestra un ejemplo de imágenes satelitales simuladas del satélite Meteosat-10 de 48 horas mediante el modelo predictivo *cy 41r2* el 1 de marzo de 2016 a las 00 UTC. Debido a la naturaleza de la órbita geoestacionaria, este producto tiene una utilidad muy limitada para los usuarios de latitudes altas; sin embargo nuestro estudio se centra en la península Ibérica y este producto es adecuado para esta tarea. La producción de estas imágenes se ejecuta como una tarea en serie, ya que es un proceso lento y consume muchos recursos. El coste computacional para generar las imágenes satelitales simuladas aumentará en el contexto del aumento en la resolución horizontal del sistema de predicción de alta resolución; estas imágenes están generadas a $T_{CO}1279$ (resolución de cuadrícula horizontal de 9 km) y 137 niveles verticales. Los errores en los datos de satélites simulados pueden derivar de errores en los parámetros de la predicción numérica del tiempo en los que se basa la simulación o la resolución del modelo entre otros. Estos datos son de dominio público pero tienen acceso restringido a cuentas académicas y científicas del ECMWF.

3

Reducción de dimensionalidad

La base matemática detrás de los algoritmos de reducción de dimensionalidad consiste en *proyectar* los datos en cierta dirección. La elección de dicha dirección se hace de forma que los datos queden plasmados en un espacio más pequeño al original perdiendo la menor cantidad de información posible. En este capítulo vamos a ver la base matemática de los algoritmos PCA, NMF e ICA; también veremos una familia de algoritmos basados en redes neuronales, los autoencoders.

Hay varias motivaciones para aplicar este tipo de técnicas a los datos. Por un lado estos algoritmos proporcionan un ahorro computacional a la hora de procesar los datos, ya que estamos reduciendo el número de atributos y la mayoría de modelos tienen un coste que depende de este parámetro. Por otro lado, nos permiten conocer mejor la «naturaleza y estructura» de nuestros datos.

3.1. PCA

El Análisis de Componentes principales (PCA) por sus siglas en inglés es una transformación lineal de los datos que manda un conjunto de observaciones a un nuevo sistema de coordenadas. En este nuevo sistema, la primera coordenada es la que tiene la mayor varianza y las que siguen están ordenadas por la cantidad de varianza retenida (es decir, en función del autovalor de la matriz de covarianzas), todas ellas bajo la restricción de ser independientes unas con otras. Estas componentes se obtienen calculando los autovalores de la matriz de covarianza de los datos; los autovectores correspondientes a los autovalores más grandes se corresponden con las nuevas direcciones del sistema de coordenadas. Cuando nuestro espacio «construido» a partir de estas direcciones tiene un número menor de coordenadas que el original estamos reduciendo la dimensión de los datos maximizando la varianza retenida.

3.1.1. Modelo

Una de las propiedades que tiene PCA es que la reconstrucción de datos a partir de la componentes con mayor varianza minimiza el error cuadrático medio [11]. Sean $\{X_i\} \in \mathcal{R}^d$; $i \in 1 \cdots n$ nuestro

conjunto de datos con dimensión d. Queremos encontrar una aplicación lineal $W \in \mathcal{R}^{m,d}$ donde $m < d$ de tal forma que dicha aplicación induzca una transformación $x \mapsto Wx$ que manda los datos originales a un espacio de dimensión menor. Deseamos también tener la aplicación lineal $U \in \mathcal{R}^{d,m}$ que se usa para devolver (aproximadamente) los datos reducidos a su dimensión original. La regla de minimización del ECM la podemos expresar de la siguiente forma:

$$\underset{W \in \mathcal{R}^{m,d}, U \in \mathcal{R}^{d,m}}{\operatorname{argmin}} \quad \sum_{i=1}^{n} \left(\|x_i - UWx_i\|^2 \right). \tag{3.1}$$

Sea (U, W) una solución de la ecuación (3.1), dichas matrices tiene una estructura concreta. Las columnas de U son ortonormales, de forma que $U^T U = I \in \mathcal{R}^{m,m}$ y $W = U^T$. Llamamos a U la pseudo-inversa izquierda de Moore-Penrose de W, mientras que W es la pseudo-inversa derecha de Moore-Penrose de U. Todo esto deriva de minimizar dicha cantidad, calculando los gradientes en función de U y W y viendo que condiciones deben de cumplir.

Basándonos en la observación anterior, podemos escribir la regla de minimización de la siguiente forma:

$$\underset{U \in \mathcal{R}^{d,m} : U^T U = I}{\operatorname{argmin}} \quad \sum_{i=1}^{n} \left(x_i - UU^T x_i \right)^2. \tag{3.2}$$

Ahora seguimos simplificando la regla de minimización con manipulaciones algebraicas básicas. Para cada $x \in \mathcal{R}^d$ y una matriz $U \in \mathcal{R}^{d,m}$ t.q $U^T U = I$ tenemos que:

$$\left(x - UU^T x \right)^2 = ||x||^2 - 2x^T UU^T x + x^T UU^T UU^T x =$$

$$||x||^2 - x^T UU^T x = ||x||^2 - tr(U^T x x^T U), \tag{3.3}$$

donde $tr()$ es el operador traza. Recordemos que tenemos un problema de optimización en función de la matriz U y debemos de minimizar cierta cantidad. Hemos visto que esta cantidad se expresa como un término que no depende de U ($|||x||^2$) y por tanto podemos omitir. El término que depende de la U es no-negativo (ya que es la traza de una matriz definida positiva) y está restando. Por tanto nuestro problema de minimización sujeto a U se convierte en un problema de maximización de este último término $tr(U^T x x^T U)$. Como la traza es un operador lineal, podemos escribir la ecuación (3.2) de la siguiente forma:

$$\underset{U \in \mathcal{R}^{d,m} : U^T U = I}{\operatorname{argmax}} \quad tr \quad U^T \cdot \sum_{i=1}^{n} \left(x_i x_i^T \cdot U \right) \left(\right. \tag{3.4}$$

3.1.2. Máxima varianza

Sea $A = \sum_{i=1}^{n} x_i x_i^T$. La matriz A es simétrica y semi-definida positiva, ya que es el resultado de multiplicar una matriz X por su traspuesta, es decir $A = XX^T$. Por tanto la podemos diagonalizar, $A = VDV^T$ donde D es una matriz diagonal y $V^TV = VV^T = I$. Los elementos de la diagonal de D son los autovalores de A y en las columnas de V están los correspondientes autovectores. Podemos asumir, sin pérdida de generalidad, que $D_{1,1} \geq D_{2,2} \geq \cdots \geq D_{d,d}$. Usando que A es semi-definida positiva, también sabemos que $D_{d,d} \geq 0$, ya que ningún autovalor es negativo. La solución para la matriz U en la ecuación (3.4) corresponde con los m autovectores de A correspondientes a los m autovalores mayores. Si tenemos las matrices «ordenadas» U serán las m primeras columnas de V. Este resultado puede parecer difícil de derivar pero en realidad se obtiene haciendo manipulaciones algebraicas básicas resolviendo el problema de optimización; omitiremos la prueba debido a su longitud.

La matriz $A = \sum_{i=1}^{n} x_i x_i^T$ se corresponde con la matriz de covarianzas de los datos (sesgados, es decir, sin restar la media); cuando hacemos su Singular value decomposition (Descomposición en autovalores y autovectores) y nos quedamos con los autovectores correspondientes a los autovalores más grandes nos estamos quedando con las direcciones que maximizan la varianza.

3.2. Otros modelos lineales

A continuación estudiaremos dos algoritmos lineales cuya finalidad es la misma que PCA. Sin embargo, el modo de obtener la representación de los datos reducida es diferente. A priori no sabemos el potencial de estos algoritmos aplicados a la reducción de imágenes y esto lo veremos en el capítulo 4, dedicado a los experimentos. En esta sección estudiaremos el fundamento teórico de cada uno.

3.2.1. ICA

ICA es un algoritmo concebido para la separación ciega de fuentes. Este proceso se fundamenta en la independencia estadística de las señales que son generadas por fuentes independientes y el algoritmo intenta reconstruir las señales originales a partir de mezclas observadas. El algoritmo ICA ha recibido gran atención en redes neuronales y procesamiento de señales, debido al potencial que tiene aplicado a dichas áreas [12].

Motivación

Imaginemos a n personas hablando simultáneamente en una sala; tenemos también distintos micrófonos distribuidos en la sala. Los micrófonos recogerán grabaciones en función del tiempo, $x_1(t)$, $x_2(t)$, $\cdots$, $x_n(t)$. Estas grabaciones serán sumas ponderadas de señales de voz, denotadas por $s_1, s_2, \cdots, s_n$. Del siguiente modo:

$$\begin{aligned} x_1(t) &= a_{1,1}s_1 + a_{1,2}s_2 + \cdots + a_{1,n}s_n \\ x_2(t) &= a_{2,1}s_1 + a_{2,2}s_2 + \cdots + a_{2,n}s_n \\ \cdots &= \cdots \\ x_n(t) &= a_{n,1}s_1 + a_{n,2}s_2 + \cdots + a_{n,n}s_n \end{aligned}$$

donde $a_{i,j}$ es la distancia de la persona i al micrófono j. Los $a_{i,j}$ conforman la denominada matriz de mezcla A. Nos interesa despejar las señales de voz de cada individuo a partir de las grabaciones de los diferentes micrófonos; este problema es conocido como «la fiesta de los cócteles». Para resolver este tipo de problemas que consisten en la separación ciega de fuentes se ha desarrollado el algoritmo ICA.

Modelo

Imaginemos que tenemos ahora un modelo donde las variables observadas $\{X\}_{i=1}^{n}$ tienen D atributos y las variables latentes $\{Z\}_{i=1}^{n}$ tienen $M < D$ atributos. Representamos las variables originales mediante la siguiente matriz:

$$X = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_D^1 \\ x_1^2 & x_2^2 & \cdots & x_D^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & \cdots & x_D^n \end{pmatrix} \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

y la matriz de variables latentes de la siguiente forma:

$$Z = \begin{pmatrix} z_1^1 & z_2^1 & \cdots & z_M^1 \\ z_1^2 & z_2^2 & \cdots & z_M^2 \\ \vdots & \vdots & \ddots & \vdots \\ z_1^n & z_2^n & \cdots & z_M^n \end{pmatrix}, \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

de modo que cada variable latente contribuirá a la construcción de la variable observada con un coeficiente. Es decir, $X_i^n = \sum_{j=1}^{M} a_{i,j} \cdot Z_j^n$. A raíz de este razonamiento, podemos expresar la matriz de mezclas de la siguiente forma:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,M} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{D,1} & a_{D,2} & \cdots & a_{D,M} \end{pmatrix} \cdot \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

De esta forma tenemos que $X = A \cdot Z$. Podemos definir una cierta A^{-1}, que llamaremos pseudo-inversa de A y revierte la mezcla de tal forma que $Z = A^{-1} \cdot X$. Pero como generalmente no conocemos la matriz A no sabemos si esta tendrá inversa; por tanto para revertir la mezcla debemos encontrar una matriz aproximada W de tal forma que el producto $W \cdot Z$ se parece mucho a la las variables originales X.

$$X \sim W \cdot Z. \tag{3.5}$$

Es importante recalcar la importancia de tener variables latentes no-gaussianas. Esta condición es necesaria para que el algoritmo ICA encuentre una solución única. Si no cumplimos lo anterior, cualquier combinación ortogonal de las variables latentes sería una solución válida. Sea K orto-normal, $(I = KK^T)$; entonces $Z' = KZ$ y $A' = AK^T$ define una solución equivalente al problema, de tal forma que $X = A'Z'$.

Fast-ICA

La variante de este algoritmo implementada en *scikit-learn*, y que por tanto es usada en los experimentos posteriores es Fast-ICA [13]. Para explicar cómo funciona este algoritmo, primero debemos de ver la versión en una unidad computacional (neuronal artificial). Tenemos un vector $\vec{w}$ el cual la neurona irá actualizando según la regla de aprendizaje proporcionada. Nos interesa encontrar una dirección (es decir, un vector unitario $\vec{w}$) de tal forma que $\vec{w}^t \cdot x$ maximiza la no-gaussianidad. Para maximizar la no-gaussianidad maximizaremos la negentropía (se usa como medida de distancia a la normalidad). Maximizaremos la siguiente cantidad:

$$J(\vec{w}^t \cdot x) \cong |E(G(\vec{w}^t \cdot x)) - E(G(v))||^2, \tag{3.6}$$

donde v es una normal estandarizada y G es una función no cuadrática; si tomamos $G(y) = y^4$, tenemos la aproximación de la negentropía por la kurtosis. Escogiendo la G de forma inteligente se consiguen aproximaciones a la negentropía mucho más precisas.

Si consideramos g la derivada de G usada en la ecuación (3.6) el algoritmo Fast-Ica para una unidad consiste en los siguientes pasos:

1.– Inicializar $\vec{w}$ aleatoriamente.

2.– $w^+ = E\{xg(w^t x)\} - E\{g'(w^t x)\}w$

3.– $w = \frac{w^+}{||w^+||}$

4.– Si no converge, volvemos al paso 2.

Recordemos que el algoritmo converge cuando los últimos dos vectores resultantes de la iteración tienen direcciones muy parecidas, es decir, su producto escalar debe ser muy cercano a 1.

Este procedimiento anterior nos estima solo una unidad o proyección de los datos. Para obtener todas las proyecciones debemos de ejecutar el algoritmo anterior usando varias unidades, con vectores directores $\{w_1 \cdots w_n\}$. Para evitar que diferentes vectores alcancen un mismo máximo, debemos de asegurarnos que las proyecciones $\{w_1^t x \cdots w_n^t x\}$ son independientes antes de cada iteración.

Una forma simple de asegurarnos que dichas proyecciones sean independientes es estimar las componentes una a una. Cuando hemos estimado p componentes, correspondientes con las direcciones $\{w_1 \cdots w_p\}$, usamos el algoritmo de un solo punto para estimar w_{p+1}. Para que esta dirección sea independiente de las anteriores, en cada iteración del algoritmo, a w_{p+1} le restamos las proyecciones de los vectores $\{w_1 \cdots w_p\}$ que ya han sido estimados previamente.

3.2.2. NMF

El algoritmo NMF consiste en la aproximación de una matriz cuyos valores son positivos o nulos mediante la multiplicación de dos sub-matrices de elementos también no-negativos que comparten una dimensión previamente determinada [14]. Se busca descomponer la matriz original X en dos sub-matrices A y B de la siguiente forma:

$$X = AB + E. \tag{3.7}$$

Todos los elementos de las matrices X, A y B son no negativos. La matriz E es el error cometido tras la factorización.

Supongamos que la matriz X tiene dimensión $l \times m$, para que la ecuación (3.7) tenga sentido la matriz A tendrá dimensión $l \times k$ y la matriz B tendrá dimensión $k \times m$. Este parámetro k son las componentes latentes de la matriz X y son la dimensión intermedia en la que se factoriza X. Se las llama componentes porque tratan de recomponer la matriz original a través de unas nuevas bases, y latentes porque no emergen hasta que el algoritmo de NMF las construye. Normalmente el número de componentes latentes se elige de modo que la suma del número de elementos en A y B sea menor que el número de elementos de X. Cuando esta última condición se cumple estamos efectuando una reducción de la dimensión de los datos.

La reconstrucción mediante las matrices A y B es una aproximación a la matriz X, perdiendo cierto grado de información.

$$AB = X^* \sim X. \tag{3.8}$$

Ahora vamos a desglosar el producto matricial para explicar un poco mejor las componentes latentes:

$$x_{ij} \sim x^*_{ij} = \sum_{k=1}^{K} a_{ik} \cdot b_{kj}. \tag{3.9}$$

Aquí se hace más visible cómo se está sumando en todas las componentes k para obtener el total. Teniendo en cuenta el carácter aditivo de la reconstrucción y que todos los elementos son no-negativos, esto indica que la visión que el NMF aporta es la de un todo compuesto por la suma de sus partes.

Para factorizar una matriz mediante NMF existen varias vías. Todas ellas están basadas en lo que en definitiva son diferentes algoritmos iterativos, cada uno con distinta medida de la distancia de la aproximación a la matriz original y reglas de actualización que mantengan la no-negatividad de los datos.

3.3. Redes neuronales

Desde el punto de vista puramente biológico, una neurona es una célula que tiene ciertas características. Cada una de estas células está compuesta por un cuerpo celular (con un núcleo) además de dos estructuras más específicas; las *dendritas* que son terminaciones de las cuales la neurona recibe estímulos eléctricos y las *terminaciones del Axón* a través de las cuales la neurona propaga un estímulo eléctrico que es dependiente de los estímulos que haya recibido por las dendritas.

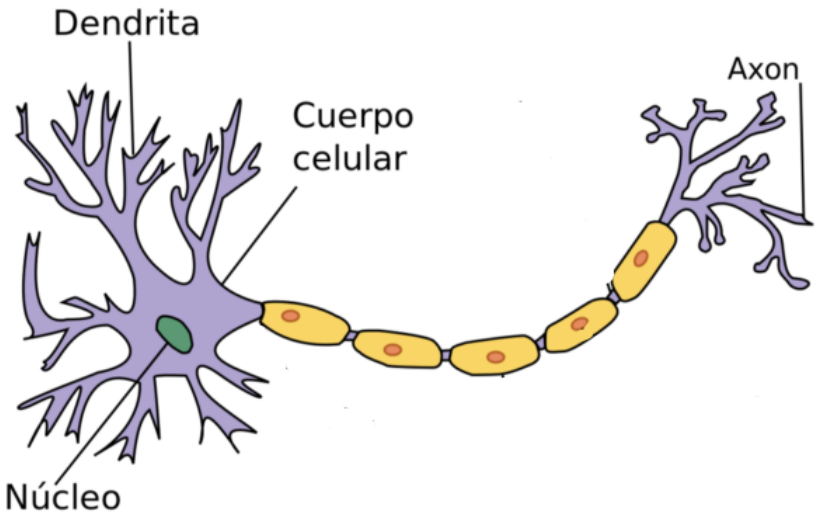

Figura 3.1: Esquema de una neurona biológica.

En una red neuronal natural (como pueden ser las de nuestro cerebro) las neuronas están conectadas entre sí y los axones transmiten el estímulo eléctrico a las dendritas de las neuronas siguientes para ir procesando la información recibida y tomar una decisión o enviar algún tipo de estímulo [15].

Un ejemplo de red neuronal natural es el *cortex visual primario* de los mamíferos. Esta es una zona del cerebro de los mamíferos muy especializada en el reconocimiento de imágenes (estáticas o dinámicas) y el reconocimiento de patrones. Los mamíferos han desarrollado especialmente esta región a lo largo de sus años de evolución y está compuesta por en torno a 140 millones de neuronas. Esta estructura es más compleja y precisa que cualquier red neuronal artificial jamás construida.

Uno de los objetivos del aprendizaje profundo (aprendizaje mediante redes neuronales con un gran número de capas ocultas) es emular a la perfección (o incluso mejorar) la toma de decisiones de un humano, sin embargo se necesita una gran capacidad de cómputo y una gran cantidad de datos de los cuales aprender.

3.3.1. Definición

Una red neuronal es un sistema computacional compuesto por unidades más simples (neuronas) que están interconectadas entre si. Las neuronas se suelen organizar por capas; la red neuronal más simple es aquella que sólo tiene una capa de entrada y una de salida. Sin embargo esquemas más generales incluyen un número arbitrario de capas ocultas con determinado número de neuronas cada una.

La organización por capas de una red neuronal funciona de tal forma que cada neurona de la capa anterior envía su salida a todas las neuronas de la capa siguiente (de esta forma se interconectan todas las neuronas de dos capas contiguas y por tanto todas las de la red). En este proyecto nos centraremos en estudiar arquitecturas de red *feed-forward*. Estas redes propagan la información que recibe en su entrada a lo largo de t instantes de tiempo; estos t instantes están relacionados con el número de capas de la red. Es decir, para cada capa se repite el mismo procedimiento y es que cada neurona de la capa recibe la suma ponderada de las salidas de la capa anterior $y_j = \sum_{i=1}^{n} w_{ij} x_i$. Posteriormente, la neurona toma ese valor y le aplica su función de activación $f(y_j)$; este valor es el que se envía a la capa siguiente.

Los pesos que conectan dos capas se suelen representar en una matriz de tamaño $n \times m$ donde n es el número de neuronas de la capa anterior y m es el número de neuronas de la capa siguiente.

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{pmatrix} \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}.$$

Aquí cada columna de la matriz representa los pesos de entrada para la neurona de la capa siguiente.

La función de salida (o de activación) suele ser una función «suave», ya que esto es conveniente para aplicar el algoritmo de retro-propagación (ya que este algoritmo está basado en el descenso por

gradiente y para ello necesitaremos calcular la derivada de la función de activación). Las más comunes son la función identidad, la función sigmoidea o la función relu (parte positiva de la entrada).

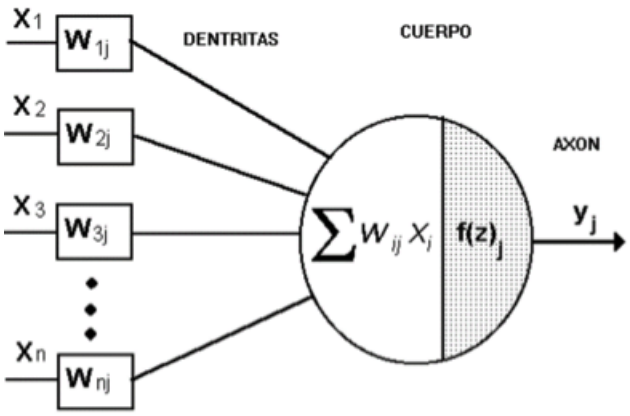

Figura 3.2: Esquema de una neurona artificial.

Estos sistemas son capaces de aprender tareas complejas sin ser explícitamente programados para ello. Para que la red aprenda se debe entrenar con un conjunto de datos. Iterando sobre este conjunto de datos se pretende minimizar cierta función de coste (error), generalmente la norma euclídea de la diferencia de la salida de la red y la salida esperada. Para ello, la red va adaptando los pesos de sus conexiones para ir minimizando el error con el algoritmo de retro-propagación. Las redes neuronales son sistemas que han demostrado funcionar muy bien para una gran cantidad de tareas diferentes; en particular las redes profundas (con un gran número de capas ocultas) son uno de los algoritmos más usados en Inteligencia Artificial actualmente.

3.3.2. Autoencoders

Un autoencoder es una red neuronal programada para aprender la función identidad. Por tanto la capa de entrada tendrá el mismo número de nodos o neuronas que la capa de salida. La función de coste será una medida del error en reconstrucción. Un autoencoder que realiza reducción de dimensionalidad de los datos tendrá una capa oculta con un número de neuronas menor que las neuronas de la capa de entrada. Esta capa con menor número de nodos la llamaremos a partir de ahora «cuello de botella» ya que es un canal estrecho a través del cual deben de pasar los datos.

Modelo lineal y equivalencia con PCA

La arquitectura de una red neuronal se basa en el número de capas ocultas y los nodos que contienen. En primer lugar vamos a hablar sobre autoencoders lineales y por tanto solo habrá una capa oculta, que será el cuello de botella y la función de activación de las neuronas de la red será la identidad.

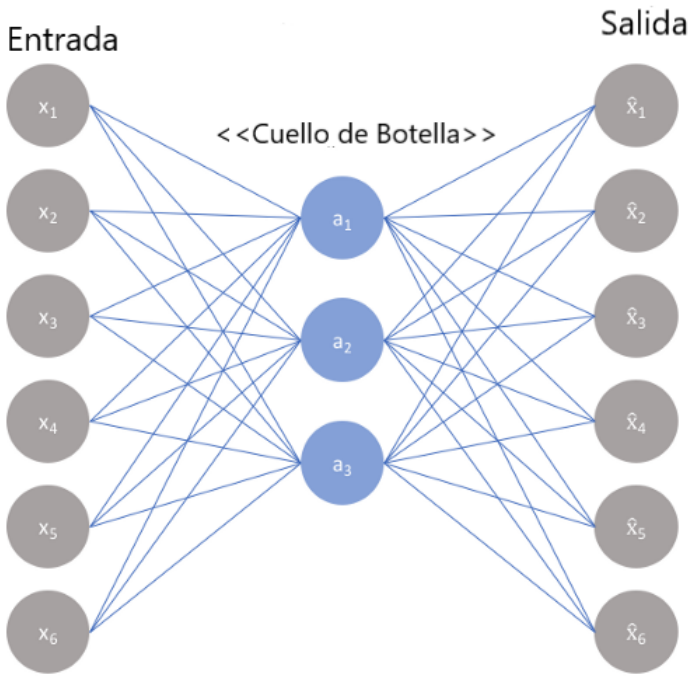

Figura 3.3: Arquitectura de un autoencoder lineal.

En este caso la compresión consiste en el envío de datos desde la capa de entrada a la capa oculta (o cuello de botella), mientras que la descompresión es el envío de datos desde la capa oculta a la capa de salida. Estas transformaciones se realizan en un paso ya que no hay capas intermedias entre el cuello de botella y la entrada/salida. En el caso de haber más capas, los procesos de compresión/-descompresión de datos son más complejos.

Si la entrada es un vector d-dimensional, digamos $X \in \mathcal{R}^{d,1}$, el resultado tras calcular la capa oculta será un vector m-dimensional $H \in \mathcal{R}^{m,1}$:

$$H = W^1 X + B^1, \tag{3.10}$$

de forma que $m < d$ para que se produzca el «cuello de botella», donde $W^1 \in \mathcal{R}^{m,d}$ es la matriz de pesos de la entrada a la capa oculta, y $B^1 \in \mathcal{R}^{m,1}$ es vector de sesgos de las unidades de entrada. La reconstrucción a partir de este vector H se efectúa de la siguiente manera:

$$Y = W^2 H + B^2, \tag{3.11}$$

donde $W^2 \in \mathcal{R}^{d,m}$ es la matriz que reconstruye los datos y $B^2 \in \mathcal{R}^{d,1}$ es el vector de sesgos de la capa oculta. El objetivo del autoencoder es encontrar unas matrices W^1, W^2 y unos vectores B^1 y B^2 adecuados de modo que minimicemos el error cuadrático medio en reconstrucción:

$$E = ||X - Y||^2 = ||X - W^2(W^1 X + B^1) + B^2||^2. \tag{3.12}$$

Cabe destacar que por simplificar la notación estamos suponiendo que nuestro conjunto de ejemplos

X es un solo ejemplo y por tanto es un vector d-dimensional. Esto se generaliza fácilmente añadiendo columnas a X, todo concuerda con las ecuaciones expuestas anteriormente. Los productos matriciales se siguen respetando y tanto B^1 como B^2 pasan a ser matrices cuyas columnas son todas iguales a la primera.

Queremos minimizar E, para ello buscamos el B^2 que minimiza E, es decir, hacemos la derivada parcial de E respecto a B^2 e igualamos a cero. Despejando esta ecuación obtenemos un cierto valor para B^2. Si ahora sustituimos dicho valor de B^2 en la ecuación (3.12) nos queda lo siguiente:

$$E = ||(X - \bar{X}) - W^2W^1(X - \bar{X})||^2. \tag{3.13}$$

Como resultado de la simplificación de los parámetros donde antes teníamos X ahora nos queda $(X - \bar{X})$, $\bar{X}$ representa la media de X. Esta diferencia representa los datos insesgados.

Este problema de optimización es muy parecido al presentado para PCA [16]; de hecho la capa oculta que representa el cuello de botella podemos pensar que es una manera aproximada de obtener los autovalores de la matriz de datos. Las matrices W^2 y W^1 son los pesos que se van a ir ajustando por descenso de gradiente para obtener un error cuadrático medio mínimo. Por tanto esperamos obtener un rendimiento parecido al obtenido por PCA utilizando un autoencoder lineal.

Modelo no lineal

El caso no lineal es una generalización del expuesto anteriormente. Como estamos reduciendo la dimensión de los datos debemos de mantener la capa correspondiente al cuello de botella (representación de los datos reducidos) en el centro de la red. La diferencia en este caso reside en que añadiremos capas adicionales entre el cuello de botella y las capas de entrada y salida. Podemos pensar que las capas que siguen a la entrada actúan como un *embudo* hacia el cuello de botella mientras las capas entre el cuello de botella y la salida actuarían como un *embudo inverso*.

3.4. Medidas de similitud

Para saber cómo de parecida es la matriz reducida a la matriz original, es necesario definir una función distancia que pueda comparar de forma directa la reconstrucción obtenida (a menor distancia con la original, mejor aproximación). Se entiende la distancia como la suma de las distancias individuales entre las matrices original y aproximada, elemento a elemento:

$$D(X^*, X) = \sum_{i,j} d(x^*_{i,j}, x_{i,j}). \tag{3.14}$$

Para medir la distancia o divergencia entre las componentes de forma individual existen diversas funciones distancia aplicables, estudiaremos las más comúnmente usadas y conocidas.

Norma euclídea

La distancia más conocida es la euclídea o norma-2.

$$D_E(X^*, X) = \frac{1}{2}\sum_{i,j}(x^*_{i,j} - x_{i,j})^2. \tag{3.15}$$

Un punto en contra de esta distancia es que elevar al cuadrado acentúa mucho las distancias grandes y disminuye las que sean menores que la unidad. Por esto, es común normalizar el conjunto de datos previamente si los datos son muy dispares, para que los resultados no se desvirtúen demasiado. Esta medida será la usada en nuestro proyecto para medir la calidad en reconstrucción; sin embargo existen muchas otras distancias que pueden ser útiles si usamos algoritmos distintos.

Distancia K-L

La distancia K-L (Solomon Kullback y Richard Leibler) aprovecha la no-negatividad de los elementos de las matrices para servirse de propiedades logarítmicas.

$$D_{K-L}(X^*, X) = \sum_{i,j}(x_{i,j} \cdot \log\frac{x_{i,j}}{x^*_{i,j}} - x_{i,j} + x^*_{i,j}). \tag{3.16}$$

Si se observa detenidamente la fórmula, si el término $x^*_{i,j} = x_{i,j}$ entonces el sumando correspondiente se anula. Esta medida de distancia suele usarse cuando trabajamos con distribuciones probabilísticas, y da una idea de la información que se pierde al tomar decisiones basándose en una distribución aproximada en lugar de la verdadera. Esta medida junto a la euclídea son las más referenciadas en artículos académicos relacionados con NMF, ya que tiene una fórmula sencilla y es fácil sacar su derivada para deducir las reglas de actualización.

Distancia I-S

La distancia I-S (s Fumitada Itakura y Shuzo Saito), fue usada inicialmente para medir la similitud (o falta de ella) de un espectro original y uno aproximado. Encaja perfectamente en los algoritmos de reducción de datos. Es muy usada para aplicaciones de audio(espectro secuencial), como separación ciega de fuentes de sonido mezcladas:

$$D_{I-S}(X^*, X) = \sum_{i,j}(\log\frac{x_{i,j}}{x^*_{i,j}} + \frac{x^*_{i,j}}{x_{i,j}} - 1). \tag{3.17}$$

4

Experimentos y resultados

En la sección anterior hemos visto el fundamento matemático de cada algoritmo estudiado, ahora vamos a comprobar qué tan eficaces son estos algoritmos aplicados a la reducción de la dimensión de imágenes. Cuando reducimos la dimensión de cualquier conjunto de datos estamos deliberadamente perdiendo información de los mismos a cambio de obtener una representación de los datos que consuma menos recursos, tanto de memoria como de coste de ejecución cuando los usamos para entrenar modelos. Esta pérdida de información la vamos a medir con el error cuadrático medio en reconstrucción.

4.1. Entorno experimental

Probaremos la eficacia de los algoritmos PCA, NMF e ICA en nuestras bases de datos de prueba. Compararemos los resultados obtenidos mediante estos métodos con un autoencoder lineal (una sola capa que es el cuello de botella y activación lineal). En el capítulo anterior vimos como la regla de optimización del autoencoder lineal es parecida a la de PCA; por tanto los resultados obtenidos mediante este método deben ser similares a los obtenidos con análisis de componentes principales. Finalmente intentaremos mejorar el error en reconstrucción usando un autoencoder no lineal (multicapa con activación relu) con una arquitectura de 3 capas manteniendo el mismo cuello de botella que en la versión lineal.

4.1.1. Hardware

Debido al elevado coste computacional que conlleva entrenar una red neuronal (en nuestro caso las arquitecturas de autoencoder lineal y multicapa), no podemos ejecutar estos experimentos en un ordenador personal. Para solucionar esto se facilitó el acceso a la computadora *eolo*, una máquina mucho más potente. Esta computadora cuenta con 40 núcleos de 2.6 GHz (2 hilos por núcleo) y una memoria RAM de 500 Gbytes. Ya que este proyecto no era el único que se estaba ejecutando simultáneamente en *eolo* hemos restringido número de núcleos que podíamos usar para paralelizar nuestras ejecuciones a 5.

Para conectar a la máquina remota desde un ordenador personal necesariamente se debe estar conectado a la VPN de la universidad. Estando en el campus esto es automático cuando nos conectamos a la red wifi *eduroam*. Sin embargo, la mayoría de las conexiones a *eolo* se han hecho desde una red wifi personal y para realizar la conexión a la VPN de la UAM se ha usado el programa *GlobalProtect*; iniciando sesión con el correo institucional de la UAM. La conexión se ha realizado mediante el comando `ssh`; de este modo accedemos a la terminal de *eolo*. Adicionalmente, para realizar una exploración inicial de los datos y mostrar resultados obtenidos lo más cómodo es trabajar con notebooks; de esta forma podemos explorar los datos de forma inmediata además de no tener que usar el comando `scp` para enviar datos a la máquina local cada vez que almacenamos los resultados de ejecución. Para trabajar con notebooks de forma remota en *eolo* debemos abrir un *tunel ssh* mediante el cual puedan fluir los datos; de este modo conectamos la máquina remota con el navegador de la máquina local. Para ejecuciones que conlleven cierto cómputo se ha usado el comando `nohup` para ejecutarlos en segundo plano de forma remota sin que perdamos todo el trabajo realizado cuando sufrimos una pérdida de conexión con el servidor.

4.1.2. Software

Todo el proyecto ha sido desarrollado usando el lenguaje de programación python. Las principales librerías usadas en este proyecto han sido las siguientes:

1.– `numpy`, para manipular los datos en forma de matrices.

2.– `pandas`, para la lectura de los datos de fichero y la manipulación de series temporales.

3.– `scikit-learn`, una librería de aprendizaje automático que contiene todos los modelos usados en el proyecto, además de otras tecnologías para el preprocesamiento de los datos y la búsqueda de hiper-parámetros (se detallará más adelante).

4.– `matplotlib`, para la generación de imágenes y gráficas.

5.– `pickle y joblib`, para la persistencia de los modelos ya entrenados.

Los algoritmos PCA, NMF e ICA están implementados en la librería scikit-learn. Para construir nuestros autoencoders hemos usado la clase `MLPRegresor` [17], que implementa el algoritmo del perceptrón multicapa.

En un proyecto de machine learning es necesario realizar un preprocesamiento de los datos a utilizar para entrenar el modelo. Un paso muy importante es la normalización de los datos (que en nuestro caso es necesario para obtener un mejor desempeño de los autoencoders). La clase usada para normalizar se llama `StandardScaler` [18]; su uso es idéntico al de un modelo, con funciones *fit* y *transform*. Hemos combinado su uso con una tecnología llamada `pipelines` [19]. Una pipeline es una clase que combina varios procesos por pasos. En nuestro caso el primer paso es la normalización de los datos y el segundo es el algoritmo usado para la clasificación. Además de esto hemos usado un `TransformedTargetRegressor` [20]; esta clase transforma el objetivo del algoritmo regresor

según cierto modelo (en nuestro caso `StandardScaler` [18], la normalización del objetivo). El algoritmo regresor que indicamos en esta clase será la pipeline antes comentada. Usar estas clases es una forma de abstraernos en cuanto al preprocesamiento de datos, es decir, no tenemos que manejar una matriz de datos normalizada y otra sin normalizar y la propia librería lo hace todo.

Finalmente, se debe elegir un parámetro de regularización (α) que maximice el rendimiento de los modelos neuronales. Para elegir el mejor α se entrenará el modelo con diferentes valores y se escogerá aquel que obtenga el mejor resultado. Para esta tarea hemos usado la clase `GridSearchCV` [21]. Esta clase realiza validación cruzada del modelo usando diferentes hiper-parámetros; además podemos paralelizar esta tarea especificando el número de núcleos que vamos a usar. Una vez se entrena el conjunto de datos con diferentes configuraciones de hiper-parámetros, identifica aquella configuración que da el mejor resultado medio y vuelve a entrenar el modelo con esta configuración (si especificamos el parámetro *refit = True*) esta vez con todos los datos disponibles. La persistencia de este objeto se realiza con la función `dump` de la librería `joblib`. Una vez lo tenemos almacenado podemos obtener el mejor clasificador ya entrenado para probar su desempeño en el conjunto de datos de validación.

El código implementado se ha subido a github [22] y en el apéndice A se explica de forma resumida la estructura del mismo.

4.1.3. Datos

Una de las bases de datos usadas es *Olivetti's Faces* [3] consistente en 400 imágenes (64×64 píxeles con valores entre 0 y 1) de 40 sujetos. Cada persona se ha tomado 10 fotografías de frente (con cierta tolerancia a giros), variando la expresión facial sobre un fondo homogéneo oscuro. La otra base de datos es *MNIST* [4]; esta es una base de datos para la clasificación y el reconocimiento de patrones ya que está compuesta por 60000 ejemplos de entrenamiento y 10000 de test. Cada imagen es un número entre el 0 y el 9, dibujado con una resolución de 32×32 píxeles. Nos interesa analizar el rendimiento de diferentes algoritmos de reducción de dimensionalidad. Por tanto, en el caso de *MNIST* nos bastará con trabajar sobre el conjunto de test. Esto es debido a que el coste computacional de los autoencoders es bastante alto y trabajando sobre un conjunto de datos más pequeño conseguimos reducir considerablemente los tiempos de ejecución.

Finalmente probaremos el rendimiento de las dos versiones de autoencoders (lineal y multicapa) en los datos del ECMWF. Estos datos consisten el las predicciones de la variable (Cloud Brightness Temperature) a lo largo del año 2020, usaremos diversas funciones de activación y compararemos la eficacia de usar esta técnica frente a PCA.

4.2. Primeros experimentos

Para fijar un número de componentes haremos un análisis de los autovalores de la matriz de covarianzas. Fijando un número de componentes principales veremos el porcentaje de varianza sobre los datos originales que estamos reteniendo y el porcentaje de reducción que estamos aplicando a los mismos. Según lo visto en 3.1.2; esta varianza sobre la información es la que retiene el algoritmo PCA cuando se queda con las n primeras componentes.

4.2.1. Olivetti's faces

Primero haremos el análisis de autovalores de los datos, para fijar el tamaño del «cuello de botella». Buscamos retener un buen porcentaje de varianza reduciendo significativamente la dimensión de los datos originales, así que fijaremos un valor acorde a esta exigencias.

En la siguiente gráfica mostramos los 4096 autovalores de la matriz de covarianzas, ordenados de mayor a menor, de esta forma podemos ver gráficamente el descenso en el valor (es decir, información) de los mismos. La línea vertical roja indica el tamaño del cuello de botella, nos quedaremos con la información a la izquierda de la linea roja. La parte que queda a la derecha de esta línea es la información que estamos perdiendo al fijar un número de componentes menor a la dimensión de los datos.

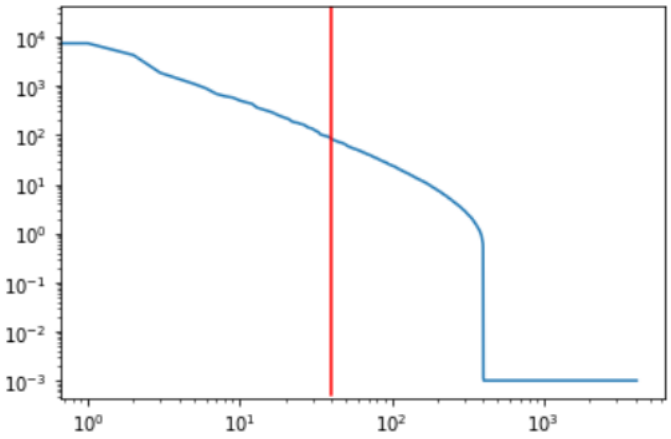

Figura 4.1: Descenso de Autovalores Olivetti.

La gráfica está representada en escala logarítmica tanto en el eje X como en el eje Y, ya que el descenso de los autovalores es bastante rápido. Usando la escala logarítmica podemos ver más detalladamente como es el descenso, sobre todo en los primeros autovalores que son los más grandes y por tanto los que mayor información retienen. La acusada bajada de la gráfica en la componente 400 se debe a que el rango de la matriz de datos es precisamente 400 (el número de caras) y por tanto una vez nos hemos quedado con ese número de componentes principales ya no añadimos información nueva.

Para esta base de datos se han escogido 40 componentes principales, lo cual corresponde a una varianza total del 90.2 % y una reducción del 99 % $= (1 - (\frac{40}{64 \cdot 64})) \cdot 100$. Se ha probado el rendimiento de los tres algoritmos estudiados en el capítulo anterior sobre estos datos con el tamaño de cuello de botella fijado. PCA e ICA tiene un rendimiento muy parecido, ambas obtienen un error cuadrático medio en reconstrucción de 0.0028. NMF obtiene un error de 0.0033, mayor que los dos algoritmos anteriores.

Mirando las reconstrucciones generadas por los tres algoritmos observamos que son todas muy similares, aunque el modo de crear las componentes principales de cada algoritmo sea distinto. PCA e ICA son algoritmos muy parecidos y tiene resultados muy similares y NMF es ligeramente peor que estos dos. Nos centraremos en los resultados obtenidos con PCA, con la finalidad de comparar las reconstrucciones de este algoritmos con diferentes arquitecturas de autoencoder.

PCA

Veremos que aspecto tienen las 5 primeras caras de la base de datos en relación a su reconstrucción. Como ya hemos comentado anteriormente hay 400 caras diferentes (de 40 sujetos), pero para ver la reconstrucción de forma gráfica solamente miraremos los 5 primeros ejemplos. Para comparar los modelos hemos medido el error cuadrático medio en reconstrucción sobre toda la base de datos.

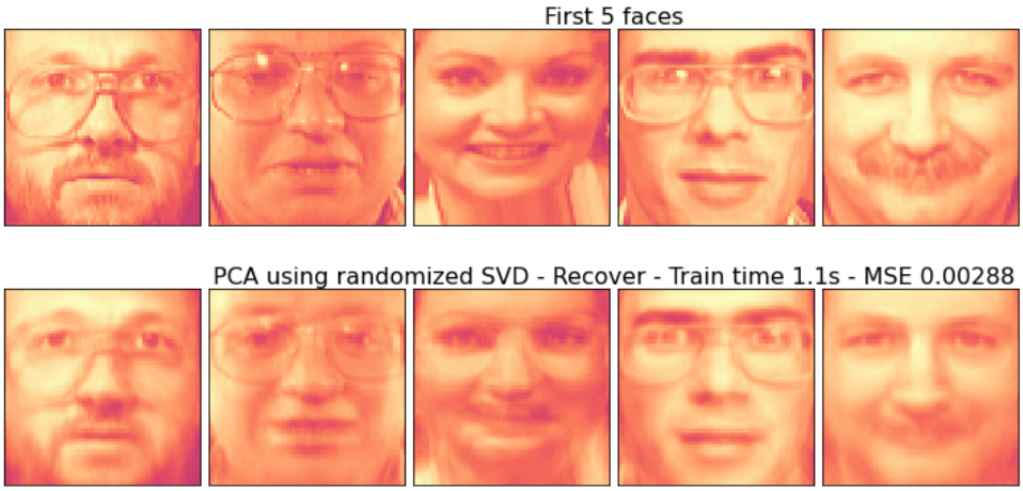

Figura 4.2: Olivetti: Originales y reconstrucción con PCA.

Podemos observar cómo la reconstrucción usando este algoritmo se parece bastante a las imágenes reales. Las imágenes reconstruidas están un poco borrosas y distorsionadas pero en general los rasgos faciales de las distintas personas son reconocibles en la reconstrucción. Por ejemplo, podemos ver que rasgos como llevar gafas, tener barba, bigote o unas cejas muy llamativas se pueden reconocer fácilmente en las imágenes reconstruidas.

Autoencoders

Probaremos a usar tanto un autoencoder lineal como uno multicapa para ver si conseguimos mejorar los resultados de los algoritmos anteriores. El cuello de botella para ambas redes será el mismo que número de componentes principales usadas para PCA, NMF e ICA (es decir, 40). En el caso del autoencoder no lineal añadiremos dos capas ocultas de tamaño 100 alrededor del cuello de botella y usaremos la función de activación «relu».

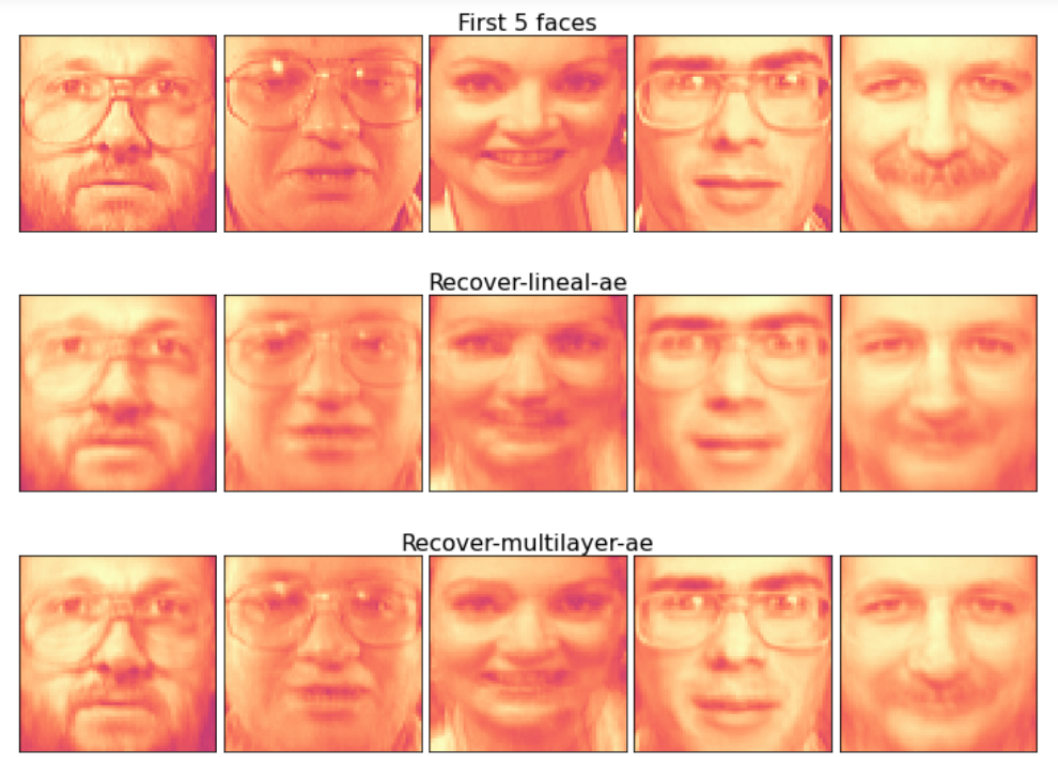

Figura 4.3: Olivetti: Originales y reconstrucción con autoencoders lineal/multicapa.

Se puede comprobar a simple vista cómo las reconstrucciones obtenidas mediante el autoencoder no lineal (multicapa) son mejores que las obtenidas mediante el lineal. Sobre todo notamos una mejora en la reconstrucción de la tercera imagen (la mujer). En general los rasgos faciales de todos los sujetos (barba, bigote o gafas) están mejor definidos en la reconstrucción obtenida mediante la arquitectura multicapa, además de poder apreciar mejor las sombras y los relieves.

El error cuadrático medio en reconstrucción del modelo lineal es 0.0029, lo cual es aproximadamente igual al obtenido mediante PCA. En el modelo multicapa el error obtenido en recostrucción es 0.0020, consiguiendo mediante esta técnica una mejora notable.

4.2.2. MNIST

Para elegir un tamaño adecuado como cuello de botella seguiremos los mismos pasos que con la base de datos anterior. En el análisis anterior hemos escogido 40 componentes principales para

retener en torno a un 90 % de varianza. En este caso, necesitamos 60 componentes principales para retener un 90.3 % de varianza y la reducción aplicada a los datos es de un 94.1 % $= (1 - \frac{60}{32 \cdot 32}) \cdot 100$.

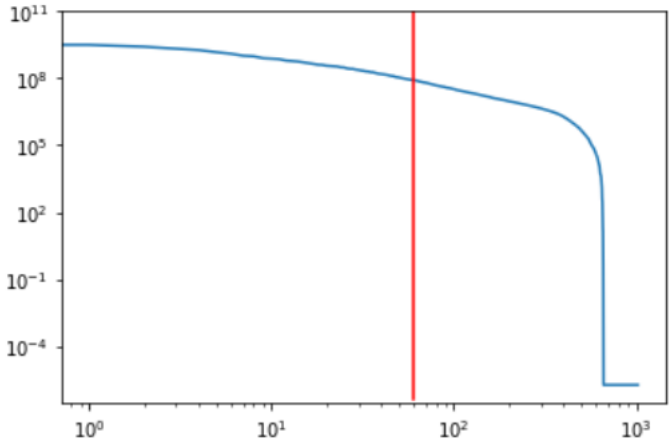

Figura 4.4: Descenso de Autovalores MNIST.

La escala tanto del eje X como del eje Y es logarítmica para observar mejor el descenso de autovalores de la matriz de covarianzas. Al igual que en el análisis anterior la linea roja vertical indica el número de componentes elegido. El descenso acusado del final de la gráfica se debe a que las imágenes de MNIST son en realidad de (28×28) píxeles a los que se le añade un *padding* de dos píxeles a cada borde, para obtener imágenes de dimensión (32×32).

PCA

En primer lugar veremos los resultados obtenidos mediante PCA. En la siguiente imagen mostramos el aspecto que tienen los datos originales y cual ha sido la reconstrucción obtenida con PCA.

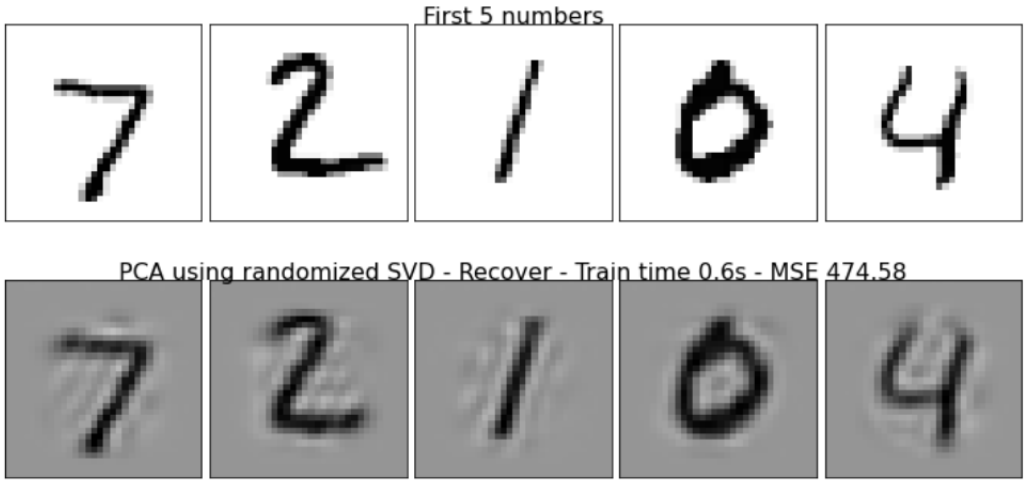

Figura 4.5: MNIST: Originales y reconstrucción con PCA.

Con 60 componentes principales obtenemos un error cuadrático medio en reconstrucción igual a 474. Se puede observar que la reconstrucción mantiene bastante bien la forma de los números, aunque

el fondo sobre el que se muestran tiene algo de ruido. Recordemos que eligiendo 60 componentes principales estamos reduciendo los datos en torno a un 94 % y aún se siguen apreciando bastante bien los patrones y características de cada número. Esto es útil para clasificar los datos usando algoritmos computacionalmente costosos como redes neuronales profundas; ya que el volumen de datos que tiene que procesar la red es un 6 % de los datos originales, reduciendo drásticamente el número de neuronas necesarias y por tanto los tiempos de ejecución.

Autoencoders

Al igual que con Olivetti, probamos a reducir la dimensión de esta base de datos usando autoencoders. Los resultados obtenidos son un error cuadrático medio de 879 con el modelo lineal y en el modelo multicapa un 365, esta última arquitectura mejora el error en reconstrucción obtenido con PCA.

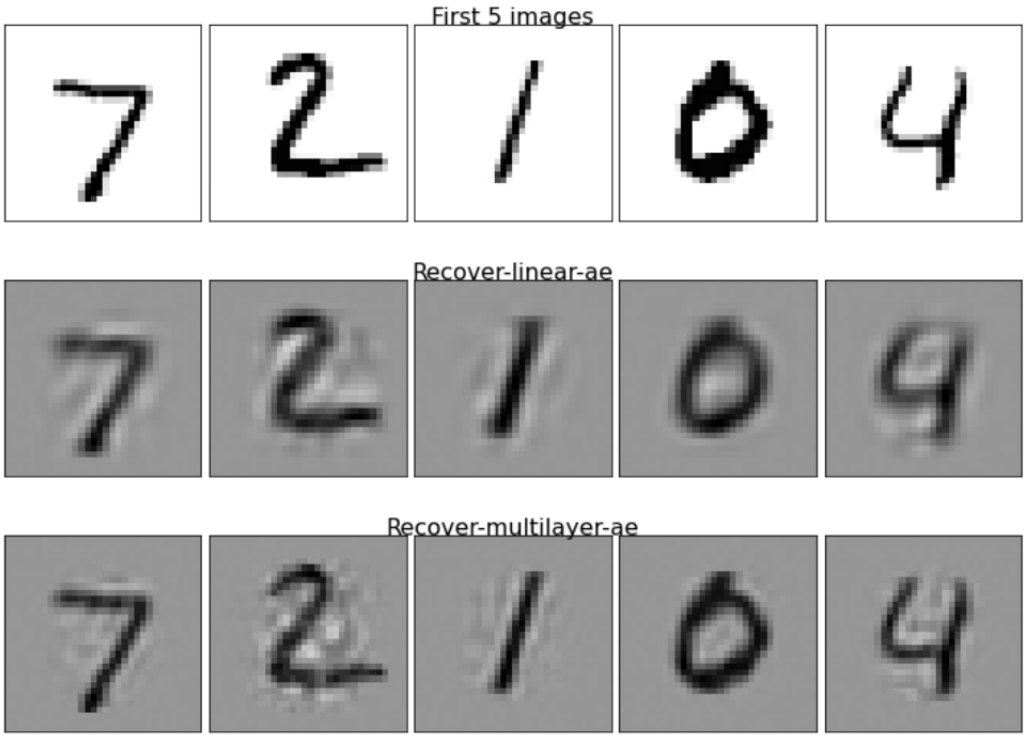

Figura 4.6: MNIST: Originales y reconstrucción con autoencoders lineal/multicapa.

Viendo la figura 4.6 podemos observar cómo las imágenes reconstruidas con el autoencoder multicapa tiene los dígitos mucho mejor definidos que las imágenes obtenidas mediante el autoencoder lineal. Comparando el modelo multicapa respecto a PCA no notamos mucha diferencia en las imágenes reconstruidas, sin embargo sabemos que la reconstrucción del autoencoder multicapa es mejor al tener un error cuadrático medio en reconstrucción menor al obtenido mediante las reconstrucciones de PCA.

4.2.3. Resumen de resultados

A continuación veremos un resumen de los resultados obtenidos en estas dos bases de datos, hemos recopilado los errores cuadráticos medios cometidos con cada algoritmo, omitiendo el estudio con NMF e ICA aplicado a MNIST. Además de los errores cometidos también reflejamos en la tabla el número de componentes elegido en cada base de datos. Con este número de componentes se ha calculado y añadido el porcentaje de reducción sobre los datos originales y el porcentaje de varianza retenido.

BBDD y cuello de botella	Var. Retenida	Reducción	AE Lineal	AE Multicapa	PCA	ICA	NMF
Olivetti's Faces (N. Comp = 40)	90,2 %	99 %	0,0029	0,002	0,0028	0,0028	0,0033
MNIST (N. Comp = 60)	90,3%	94 %	879	365	474	---	---

Tabla 4.1: Resumen de errores en reconstrucción MNIST y Olivetti.

Como podemos observar, en ambas bases de datos hemos elegido un número de componentes (cuello de botella) que suponga una retención de varianza de 90 % aproximadamente. Vemos en ambos casos como el uso de un autoencoder con arquitectura multicapa es el mejor modelo para obtener menor error cuadrático medio en reconstrucción, mejorando el desempeño de PCA. En la base de datos *Olivetti* el error de PCA es aproximadamente igual al error del autoencoder lineal, sin embargo en *MNIST* el autoencoder lineal es considerablemente peor (en términos de reconstrucción) que PCA.

4.3. Experimentos con datos de satélite

Teniendo una idea general de cómo funcionan los algoritmos de reducción de dimensionalidad explicados en el capítulo 3 aplicados a las bases de datos *Olivetti* y *MNIST*; vamos a aplicar estas técnicas a los datos de satélite simulados del ECMWF [10] y mediremos la calidad de las reconstrucciones obtenidas con cada algoritmo.

Posteriormente se programará un conjunto de modelos para predecir la evolución temporal de estos datos. Mediremos la calidad de la predicción usando las imágenes originales y también la representación de las mismas en el *cuello de botella* con diferentes modelos.

4.3.1. Reducción de dimensionalidad

En primer lugar haremos el correspondiente análisis de la matriz de covarianzas de los datos (2019). Estos datos tienen una resolución temporal de tres horas y corresponden a un año entero, por tanto habrá 2920 imágenes. Cada imagen se centra en la península ibérica y tiene un tamaño de $69 \times 85 = 5865$ píxeles.

Para obtener una mejor resolución temporal aplicamos interpolación lineal a los datos originales,

de modo que tengamos una imagen por cada hora. De este modo conseguimos 8760 imágenes.

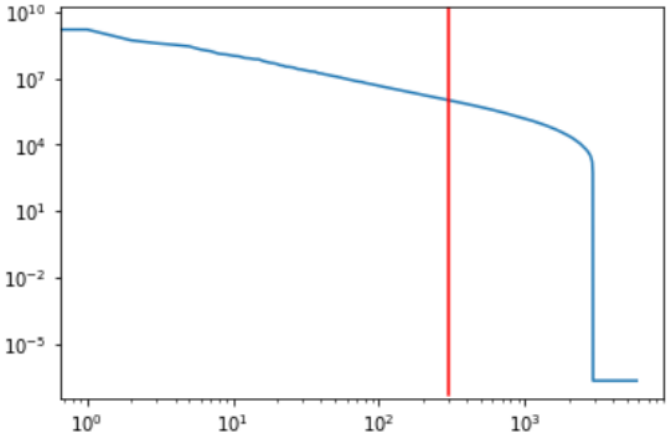

Figura 4.7: Descenso de Autovalores SSD.

Hemos fijado el cuello de botella a 300 componentes, ya que con este número conseguimos retener un 96 % de varianza y estaríamos aplicando una reducción del 94.1 %. Podemos apreciar cómo a partir de las 2920 componentes la gráfica representada en la figura 4.7 hace una bajada muy notable; esto es debido a la interpolación, a partir de las 2920 componentes ya no hay información nueva.

PCA

Primero vamos a estudiar el efecto que tiene PCA aplicado a estos datos. Usaremos los datos correspondientes a 2019 para entrenar el modelo y posteriormente mediremos la calidad en reconstrucción tanto de los datos de entrenamiento como los datos correspondientes al año 2020 (test).

Con este algoritmo obtenemos un error cuadrático medio en reconstrucción de 3,4 en el conjunto de train (2019) y un 8,9 en el conjunto de test (2020). En la siguiente figura podemos ver es aspecto que tienen las 3 primeras imágenes (correspondientes a las 3 primeras horas de cada año) y su reconstrucción.

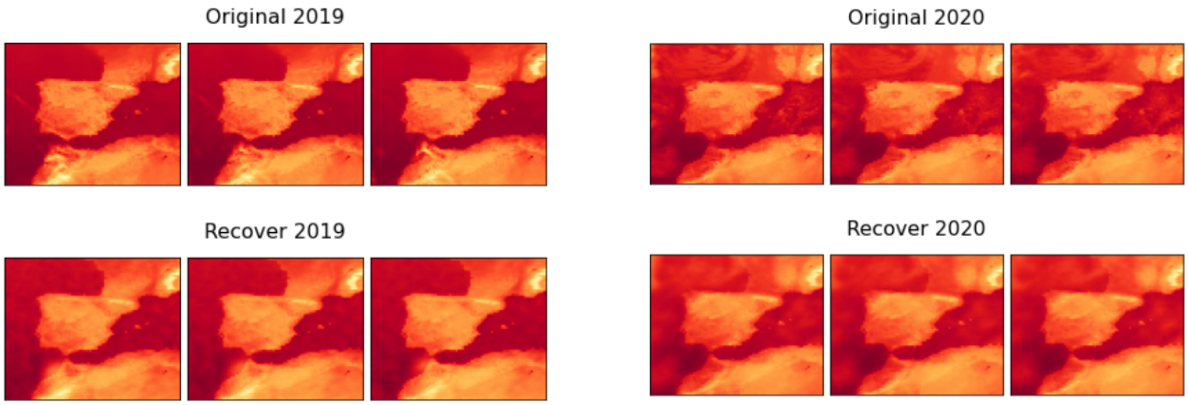

Figura 4.8: Tres primeras horas de 2019 / 2020 y su reconstrucción usando PCA.

Lo que observamos en primer lugar es que ambas reconstrucciones son similares al original; son

fácilmente reconocibles tanto la península ibérica como la costa norte africana.

Con los datos correspondientes a 2019 se ha entrenado el modelo, por tanto, esperamos que estas reconstrucciones sean muy parecidas al original. Observamos que efectivamente son muy parecidas, habiendo alguna discrepancia sobre todo en el norte de África y si nos fijamos con mucho detalle podremos observar el fondo un poco difuminado. El tiempo atmosférico de este día (1 de enero de 2019) era soleado ya que no observamos cobertura de nubes en los datos originales.

Los imágenes mostradas de 2020 corresponden (al igual que los de 2019) a las tres primeras horas del 1 de enero de ese año. Podemos observar cómo hay cobertura de nubes en los datos originales y la reconstrucción con PCA y si nos fijamos en las nubes están algo difuminadas en reconstrucción. Como estos datos son los de test cabe esperar que el rendimiento de PCA sea peor que con los datos de 2019. En las reconstrucciones mediante PCA de ambos años se distingue perfectamente tierra de mar y sorprende la calidad en reconstrucción si observamos las costas.

Para realizar una comparación de forma más cómoda de las reconstrucciones a lo largo del tiempo se ha implementado una función que crea animaciones con las imágenes originales y reconstruidas. Esto se detalla en el apéndice A.

Autoencoder lineal

En primer lugar implementaremos un autoencoder lineal con un cuello de botella de 300 neuronas. Hacemos uso de la clase `MLPRegressor` que implementa un perceptrón multicapa.

```
21 mlp = MLPRegressor(hidden_layer_sizes = (300),
22                             solver = 'adam',
23                             activation = 'identity',
24                             random_state=1,
25                             tol = 1.e-6,
26                             n_iter_no_change = 100,
27                             max_iter=1000,
28                             shuffle = True,
29                             early_stopping = False,
30                             verbose = True)
31
32 regr = Pipeline(steps=[('std_sc', StandardScaler()),
33                         ('mlp', mlp)])
34
35 y_transformer = StandardScaler()
36 inner_estimator = TransformedTargetRegressor(regressor=regr,
37                                               transformer=y_transformer)
```

Figura 4.9: Implementación del autoencoder lineal en python.

Para la arquitectura lineal usaremos la función de activación identidad y la única capa oculta será el cuello de botella. Para la elección del *solver* teníamos varias opciones; *Stochastic gradient descent (SGD)*, *Adaptive Moment Estimation (ADAM)* [23] o *métodos newtonianos*. Tras una prueba con cada método elegimos *ADAM* por que este proporcionaba mejores resultados. El parámetro `n_iter_no_change` indica el número de épocas que dejaremos trabajar al perceptrón cuando

no se mejora la tolerancia; en este caso la tolerancia es 10^{-6}, un valor muy pequeño y por tanto pararemos el entrenamiento del perceptrón cuando los resultados empeoran durante 100 épocas seguidas. Se puede observar también en la figura 4.9 como se configura un `pipeline` y un `TrasnsformedTargetRegressor` para la normalización automática tanto de la entrada como del objetivo.

Los resultados obtenidos con este método han sido muy parecidos a los de PCA, teniendo un error cuadrático medio (ECM) en reconstrucción de 3,5 en los datos de entrenamiento (2019) y un ECM de 9,2 en los datos de test (2020).

Autoencoder multicapa

Una vez hemos hecho las pruebas con la arquitectura lineal nos disponemos a crear un modelo con más capas manteniendo el cuello de botella de 300 neuronas. Para la creación de este modelo añadiremos dos capas adicionales de 1000 neuronas que actuarán como *embudo* hacia el cuello de botella y *embudo inverso* hacia la salida (desde el cuello de botella). Para la implementación en python, será idéntica a la mostrada en la figura 4.9; solo que en el parámetro `hidden_layer_sizes` indicaremos la tupla (1000, 300, 1000). Además, la función de activación ya no será la identidad y usaremos relu.

Obtenemos resultados peores a los obtenidos con la arquitectura lineal y PCA. En entrenamiento tenemos un ECM en reconstrucción de 7,8 y en test un ECM de 20,4. Destacamos que con la arquitectura planteada (entrada y salida de 6000 neuronas, además la configuración de (1000, 300, 1000) en las capa oculta) tenemos un total de 12,6 millones de pesos para ajustar en entrenamiento. Aunque disponemos de un computador bastante potente esta cantidad de parámetros ajustables es muy elevada y puede resultar en problemas computacionales.

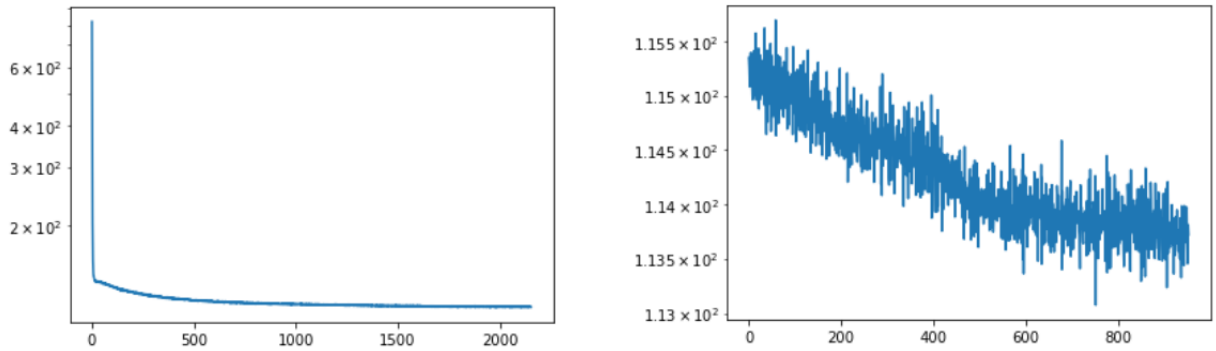

Figura 4.10: Curva de aprendizaje del autoencoder multicapa y detalle de las últimas 1000 épocas.

Analizamos la curva de aprendizaje del autoencoder multicapa, mostrada en la figura 4.10. El modelo se ha entrenado con un parámetro `n_iter_no_change` igual a 200 y 5000 iteraciones máximas; entrenar estos modelos conlleva un tiempo de en torno a una semana (hiperparametrizando el alpha), por tanto necesitamos una gran cantidad de recursos computacionales. Se puede observar

como a partir de la época número 1000 la mejora es inapreciable (la escala del eje Y es logarítmica). Sin embargo, si analizamos con más detalle las últimas 1000 épocas vemos como sigue mejorando ligeramente. Se deja como trabajo futuro entrenar esta arquitectura con máquinas más potentes.

Resumen de errores en la reconstrucción

Veremos ahora una tabla con el resumen de los errores en reconstrucción de los diferentes algoritmos de reducción de dimensionalidad aplicados a datos de satélite simulados.

Algoritmo de Reducción	ECM train (2019)	ECM test (2020)
PCA	3,4	8,9
Autoencoder Lineal	3,5	9,2
Autoencoder Multicapa	7,8	20,4

Tabla 4.2: Errores cuadráticos medios en reconstrucción SSD.

EL resultado obtenido con el autoencoder lineal entra dentro de nuestras espectativas; siendo muy parecido al obtenido con PCA. Sin embargo el resultado del autoencoder con la arquitectura multicapa no mejora los resultados previos (algo que si hemos observado con las bases de datos *Olivetti* y *MNIST*). Concluimos que el método óptimo para reducir la dimensión de estos datos con los recursos computacionales disponibles es PCA.

4.3.2. Predicción de la evolución temporal

Ya hemos estudiado cómo actúan los distintos algoritmos de reducción de dimensionalidad aplicados a los datos de satélite. Ahora vamos a aplicar diferentes métodos de predicción de la evolución temporal tanto a las imágenes originales como a su representación en el cuello de botella (es decir, sobre los datos comprimidos). Disponemos de datos con una resolución temporal de 3 horas, para este estudio no tiene sentido que apliquemos interpolación lineal ya que nos interesa saber que pasará en las tres horas siguientes.

Para realizar las predicciones tendremos que «desplazar» los datos una posición hacia delante (datos futuros) e intentar obtenerlos a partir de los datos originales aplicando algún algoritmo de predicción. Para obtener la representación en el cuello de botella hemos usado PCA, ya que es complejo obtener los datos reducidos en el caso de los autoencoders con la librería usada en este proyecto. Además de ser PCA el algoritmo que mejor resultados obtiene; según hemos visto en 4.3.1.

Modelo de persistencia

En primer lugar nos fijamos en el error del modelo de persistencia. Este modelo consiste en no aplicar ningún tipo de predicción a los datos y comparar directamente la imagen actual con la correspondiente a las siguientes tres horas (como si la evolución temporal fuese estática).

Obtenemos un error cuadrático medio entre los datos pasados y futuros de 92,8 en el año 2019 y de 92,5 en 2020. No sorprende que este error sea tan alto, ya que al fin y al cabo estamos comparando dos instantes de tiempo diferentes; esta cifra nos pone en contexto de la discrepancia que hay entre los datos pasados y futuros y será el objetivo a batir con los siguientes algoritmos. Si medimos el resultado obtenido con el modelo de persistencia sobre la representación en el cuello de botella comprobamos que el error es menor, de 83,2 en el año 2019 y de 81,2 en 2020. Este resultado sorprende ya que a priori podríamos pensar que al perder información las discrepancias iban a ser mayores sin embargo esto no ocurre.

Predictor ridge

El predictor ridge es un modelo lineal para la regresión. Es un algoritmo idéntico a la regresión lineal pero añadiendo un término de regularización. Para hacer este estudio hemos hiper-parametrizado este parámetro de regularización (alpha).

Usando un predictor lineal con el mejor alpha obtenido del correspondiente análisis de hiper-parámetros observamos que reducimos el error prácticamente a la mitad obteniendo un ECM en 2019 (datos de entrenamiento tanto del modelo ridge como de PCA) de 44,8 en datos originales y 46,4 en la representación en el cuello de botella. En el año 2020 (datos de test) obtenemos un ECM de 51,4 sobre los datos originales y de 53,8 en la representación en el cuello de botella. Esto es esperable ya que siempre obtendremos mejor rendimiento en los datos de entrenamiento que en los de test; además observamos como la predicción usando los datos originales es ligeramente mejor que la realizada con los datos comprimidos, lo cual es también esperable ya que con los datos originales estamos usando toda la información disponible y con la representación en el cuello de botella estamos perdiendo parte de esta información.

Predictor neuronal

Para mejorar el rendimiento de la predicción lineal vamos a usar un perceptrón multicapa. Es el mismo algoritmo usado para implementar los autoencoders de la sección anterior pero en este caso se van a usar para la predicción. La entrada del modelo será en este caso los datos actuales e intentaremos obtener los correspondientes a las siguientes tres horas (este será el objetivo con el que se comparará la salida). Se han probado diversas configuraciones de capas ocultas, descartando aquellas con capas ocultas de tamaño muy reducido en comparación con la entrada (ya que no queremos perder demasiada información) y concluimos que la mejor opción es usar dos capas ocultas de 1000 neuronas con activación relu, para mantener un equilibrio entre un coste computacional asumible y una pérdida de información no muy acusada. Además, como en el caso del modelo lineal hemos hiper-parametrizado el parámetro de regularización, eligiendo el modelo con mejores resultados.

Usando los datos originales hemos tenido problemas de convergencia del modelo (principalmente

debido a la gran cantidad de pesos que supone una entrada y salida del orden de 6000 neuronas); esto ya nos pasó en la sección anterior cuando decidimos usar el algoritmo del perceptrón para implementar un autoencoder multicapa con dos *embudos* de 1000 neuronas. Sin embargo, el resultado verdaderamente valioso de este análisis ha sido el aplicar este predictor a la representación reducida de estos datos. Obtenemos un ECM de 23,2 en los datos de entrenamiento y un 49,3 en los datos de 2020; produciendo una mejora considerable respecto al modelo lineal.

Resumen de resultados en la predicción

Mostramos ahora el resumen de resultados con todos los modelos estudiados; cada fila representa un modelo distindo. La dos primeras columnas son el resultado de la predicción sobre los datos originales y las dos siguientes la predicción usando la representación en el cuello de botella de los mismos (datos reducidos). Con los datos del año 2019 hemos entrenado tanto el clasificador usado como PCA y usamos los datos de 2020 como test (por tanto esperamos que los errores obtenidos en la predicción de este año sean mayores).

Modelo	ECM 2019	ECM 2020	ECM 2019 (Reducidos)	ECM 2020 (Reducidos)
Persistencia	92,8	92,5	83,2	81,2
Predictor Lineal (Ridge)	44,8	51,4	46,4	53,8
Predictor Neuronal (MLP)	50,8	56,9	23,2	49,3

Tabla 4.3: Resumen de resultados en predicción de la evolución temporal.

Lo primero que observamos es que todos los modelos mejoran el error del modelo de persistencia. Los errores cuadráticos medios del modelo neuronal sobre los datos originales son mayores que los obtenidos mediante el modelo lineal (debido al problema de convergencia del modelo ya comentado anteriormente). Sin embargo, destacamos la mejora obtenida aplicando el modelo neuronal sobre la representación en el cuello de botella; estos errores son menores que los obtenidos mediante el modelo lineal tanto usando los datos originales como su representación comprimida.

5

Conclusiones y trabajo futuro

Tras el trabajo realizado podemos sacar una serie de conclusiones. En primer lugar destacamos que hay un gran margen de compresión en imágenes de satélite; es decir, podemos reducir bastante la resolución de estas imágenes sin perder demasiada información. En nuestro proyecto hemos reducido la dimensión un 95 % aproximadamente y obtenemos un ECM bastante bajo en relación con la escala de los datos. La variable estudiada son grados (medidos en Kelvin) y por tanto la magnitud de los mismos oscila entre 160-190 aproximadamente; reduciendo drásticamente el número de atributos de los datos obtenemos un ECM en reconstrucción de 3,2 con PCA que puesto en perspectiva con la escala de los datos es bastante aceptable (considerando que hemos comprimido los mismos un 95 %).

En segundo lugar nos hemos dado cuenta que el entrenamiento de redes neuronales con muchas neuronas implica la actualización de una gran cantidad de pesos y por tanto es un proceso computacionalmente muy costo; para la realización de este trabajo disponíamos del computador *eolo* pero aún así hemos tenido problemas de convergencia con los modelos multicapa tanto en el autoencoder como en la predicción de la evolución temporal. Sabemos que el modelo de autoencoder multicapa es más efectivo sobre imágenes (según lo estudiado en las bases de datos *Olivetti* y *MNIST*); sin embargo por la gran cantidad de pesos entrenables en imágenes de satélite no hemos conseguido que converja a una solución que mejore la compresión mediante PCA.

Finalmente, en los experimentos realizados de predicción de la evolución temporal hemos obtenido mejores resultados con la representación reducida de las imágenes que con las imágenes originales usando el modelo neuronal; además de un ahorro computacional muy notable. Usar los datos reducidos nos ha solucionado parcialmente el problema de convergencia comentado anteriormente y podemos obtener resultados mucho más rápido que trabajando con los datos originales.

5.1. Trabajo futuro

Se deja como trabajo futuro estudiar con más detalle la convergencia de los modelo neuronales con más recursos computacionales. Pensamos que estos modelos necesitan una cantidad de épocas varios órdenes de magnitud superiores a los que se han explorado en este proyecto (por falta de

recursos computacionales y tiempo). Aumentar el parámetro `n_iter_no_change` o la paciencia en entrenamiento ha mejorado notablemente el desempeño en las bases de datos *Olivetti* y *MNIST* por tanto esperamos que ocurra lo mismo con datos de satélite.

Es interesante para proyectos futuros estudiar también mejores métodos para la predicción de la evolución temporal; usando tanto los datos originales como reducidos. En este proyecto no ha sido el tema central pero tiene aplicaciones muy interesantes y la predicción del instante posterior conociendo el anterior es un preámbulo para el estudio de métodos capaces de predecir varios instantes de tiempo posteriores (llegando a días completos o incluso semanas). Usando los datos comprimidos (a un 95 %) se puede proporcionar como entrada a la red los datos de una gran cantidad de instantes de tiempo anteriores sin que esto suponga serios problemas computacionales. Estos datos comprimidos retienen una gran cantidad de información y para la predicción de instantes de tiempo futuros pensamos que puede ser útil usar esta configuración.

Bibliografía

[1] P. web del EUMETSAT (Link).

[2] P. web del ECMWF (Link).

[3] B. de Datos, "Olivetti's faces," (Link).

[4] B. de Datos, "Mnist," (Link).

[5] P. web del producto SEVIRI (Link).

[6] A. Catalina Feliú and J. R. Dorronsoro Ibero, "Data analytics en energias renovables: organización y análisis de medidas satelitales de radiación," *UAM (TFG)*, pp. 1–6, 2016.

[7] P. web del Data Centre de EUMETSAT (Link).

[8] ECMWF-Research *ECMWF*, 2011. (Link).

[9] R. Buizza, "Chaos and weather prediction," *ECMWF*, pp. 1–5, January 2000.

[10] C. Lupu and T. Wilhelmsson, "A guide to simulated satellite images in the ifs," *ECMWF*, pp. 1–10, 2016.

[11] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press., first ed., 2014.

[12] D. A. Alvarez and E. Giraldo, "Ica applied to image feature extraction," *ISSN*, pp. 1–6, 2008.

[13] A. Hyvärinen and E. Oja, "Independent component analysis: Algorithms and applications," *Helsinki University of Technology*, pp. 1–16, 2000.

[14] A. Cichock, S. Cruces, and A. Shun-ichi, "Generalized alpha-beta divergences and their application to robust nonnegative matrix factorization," *ISSN*, pp. 134–138, 2011.

[15] D. Díaz Vico and J. R. Dorronsoro Ibero, "Deep neural networks," *UAM (TFM)*, pp. 1–33, 2012.

[16] E. Plaut, "From principal subspaces to principal components with linear autoencoders," pp. 1–5, December 2018.

[17] MLPRegressor, "Scikit-learn," (Link).

[18] StandardScaler, "Scikit-learn," (Link).

[19] Pipeline, "Scikit-learn," (Link).

[20] TransformedTargetRegressor, "Scikit-learn," (Link).

[21] GridSearchCV, "Scikit-learn," (Link).

[22] J. M. Chacón Aguilera, "Código implementado para el proyecto," *Github*, 2021. (Link).

[23] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *ICLR*, pp. 1–3, 2015.

Apéndices

A

Estructura del código y creación de animaciones

El código implementado para la realización de este proyecto está publicado en github [22]. Hemos organizado el código en dos directorios principales. En el directorio llamado *DimReduction* hemos realizado los experimentos en las bases de datos *Olivetti* y *MNIST*. En el directorio llamado *SSD* se han realizado los experimentos relacionados con datos de satélite. Adiccionalmente en el repositorio hemos incluido algunas de las referencias bibliográficas usadas.

Estos dos directorios siguen la misma estructura, dentro de los mismos podemos encontrar uno o varios archivos formato notebook donde se muestran los resultados obtenidos con cada base de datos, además de ficheros .py donde se definen las funciones usadas y también los *scripts* para la ejecución en remoto. En la carpeta *stored_objs* de ambos directorios hemos guardado los modelos ya entrenados.

Adicionalmente, se ha programado una función que crea animaciones (en formato gift) con las imágenes originales y reconstruidas mediante PCA. Esto lo hemos hecho para poder apreciar de forma más cómoda cómo es la reconstrucción durante un periodo de tiempo concreto. En la carpeta *gitfs* dentro del directorio *SSD* se encuentran dos animaciones correspondientes a los datos del 1 de enero de 2019 (originales y reconstruidos). La función que crea las animaciones está definida en el archivo `ssd_view.py`. Dicha función recibe los datos en formato dataframe, el número de componentes del cuello de botella, la hora de comienzo de animación, lo hora de fin de animación y opcionalmente se puede especificar los fraps por segundo (fps); que por defecto está configurado a 2. Esta funcionalidad fue bastante útil en el proceso de buscar un tamaño para el cuello de botella, eligiendo aquel que retenía una buena cantidad de información y que también producía unas animaciones muy parecidas a las de los datos originales.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID